

БИБЛИОТЕЧКА
ПРОГРАММИСТА

В. А. ЕВСТИГНЕЕВ

Применение теории графов в программировании



БИБЛИОТЕЧКА ПРОГРАММИСТА

В. А. ЕВСТИГНЕЕВ

ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ

Под редакцией А. П. ЕРШОВА



МОСКВА «НАУКА»

ГЛАВНАЯ РЕДАКЦИЯ

ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ

1985

22.18

Е 26

УДК 519.6

Евстигнеев В. А. Применение теории графов в программировании./Под ред. А. П. Ершова.— М.: Наука. Главная редакция физико-математической литературы, 1985—352 с.

Книга посвящена вопросам использования методов теории графов для исследования структуры сложных программ, определения их параметров, верификации, организации хранения и поиска информации, распределения памяти и для решения других вопросов, возникающих в системном программировании и смежных областях.

Владимир Анатольевич Евстигнеев

ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ

Серия «Библиотечка программиста»

Редактор *Н. В. Полякова*

Техн. редактор *В. Н. Кондакова*

Корректоры *Т. С. Плетнева, М. Л. Медведевская*

ИБ № 12380

Сдано в набор 16.05 84. Подписано к печати 15.01.85. Т-01212. Формат 84×108¹/₃₂. Бумага тип. № 3. Обыкновенная новая гарнитура. Высокая печать. Усл. печ. л. 18,48. Усл. кр.-отт. 18,69. Уч.-изд. л. 18,96. Тираж 20 000 экз. Заказ № 223. Цена 1 р. 20 к.

Издательство «Наука»

Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

4-я типография издательства «Наука»

630077 г. Новосибирск, 77, ул. Станиславского, 25

Е $\frac{1702070000-029}{053(02)}-85$ 37—84

© Издательство «Наука».
Главная редакция
физико-математической
литературы, 1985

ОГЛАВЛЕНИЕ

От редактора	5
Предисловие	7
Глава 1. Основные понятия	9
§ 1. Основные определения теории графов	9
§ 2. Графы как модели программ, данных и процессов	19
§ 3. Графы как объекты обработки информации	26
Библиографический комментарий	32
Глава 2. Глобальный анализ графов	33
§ 1. Нумерации, выявляющие логическую структуру графа	33
§ 2. Логический анализ управляющих графов. Линейные компоненты и компоненты сильной связности	52
§ 3. Гамаки, полугамаки и шлейфы	76
§ 4. Интервалы и сводимые графы	83
§ 5. Контуры в орграфах	105
Библиографический комментарий	118
Глава 3. Итеративные алгоритмы глобального анализа графов. Пути и покрытия	121
§ 1. Итеративный алгоритм Килдала	121
§ 2. Пути в орграфах	130
§ 3. Пути, удовлетворяющие дополнительным ограничениям. Покрытия	138
§ 4. Отыскание доминаторов в орграфе	150
Библиографический комментарий	165
Глава 4. Оптимизационные задачи на графах	166
§ 1. Построение оптимальных нумераций	166
§ 2. Конструирование оптимальных деревьев	192
§ 3. Балансированные деревья	214
Библиографический комментарий	239

Глава 5. Разрезания и раскраска графов	241
§ 1. Разрезания графов	241
§ 2. Раскраска графов	278
Библиографический комментарий	292
Глава 6. Применение теории графов в программировании	294
§ 1. Анализ и тестирование программ. Вычисление характеристик программ	294
§ 2. Применение методов теории графов к организации вычислительного процесса	309
§ 3. Применение деревьев для организации больших массивов информации	325
Библиографический комментарий	340
Список литературы	342
Предметный указатель	350

ОТ РЕДАКТОРА

Программирование — это новый вид универсальной деятельности. Человек должен вложить в ЭВМ все, что видит, слышит, знает, и научить ее всему, что делает сам. Об этом приходится время от времени вспоминать, для того чтобы не суживать чрезмерно кругозор при занятии программированием. М. Нива в своей научной переписке как-то указал, что над многими информатиками довлеет наивное заблуждение, связанное с кажущейся простотой основных структур программирования: им кажется, что сравнительно небольшое количество универсальных методов позволяет им найти ключ к решению практически любой «реальной» задачи программирования. Сходную мысль высказывал Э. Дейкстра: воюя с теми, кто оценивает продуктивность программиста по числу написанных команд, он отмечал, что при грамотном программировании на тысячу строк программного текста надо написать в десять раз больше рассуждений и доказательств, гарантирующих всеобщую применимость программы.

Все три высказанных тезиса некоторым образом фокусируются на материале этой книги. Важнейшим свойством информационной модели или управляющей системы является ее структура, или, говоря математическим языком, совокупность бинарных отношений на наборах элементарных единиц данных и действий. Эти структуры данных и структуры действий являются по выражению А. А. Берса единственными ипостасями программ и обрабатываемой ими информации, в которых они могут существовать в воображении программиста и во чреве компьютера.

Вот почему графы являются основной конструкцией для программиста. Графы обладают огромной, неисчерпаемой изобразительной силой, соразмерной масштабу задачи программирования.

Поток литературы по теории графов сейчас нарастает с экспоненциальной скоростью. Как и всегда в истории науки, этот поток двоякий: с одной стороны, в нем наблюдается тенденция к выделению струй с помощью формулирования фундаментальных свойств, выражающих общую природу графов или их крупных классов; с другой стороны, эти струи дробятся и образуют сплетения, отражающие информационное богатство объекта и его теории.

Из всего сказанного вытекает простое следствие: программисту о графах нужно много знать, при этом с большим запасом по отношению к любой конкретной задаче.

Предлагаемая вниманию читателя книга является одной из первых попыток очертить некоторый запас знания по теории графов, необходимый для программиста. Хотелось бы подчеркнуть, что книга призвана не столько завершить, сколько открыть период интеграции знания по теории графов и её приложений к программированию в виде книг и руководств. Наиболее серьезное ограничение книги — это, по существу, охват лишь тех программных структур, которые сложились в модели последовательных вычислений. Другое обстоятельство, которое характеризует текущий момент скорее как период становления, нежели как период завершения, — это концентрация собственно программистской проблематики в последней главе. Главный объем книги посвящен обзору алгоритмов и свойств, взятых из журнальной литературы. Технологическое погружение в программные процессоры, принципы отбора и ряд других моментов, придающих знанию рецептурный характер, еще ждут своего исследования. Эти ограничительные оговорки, однако, лишь подчеркивают важность и актуальность материала, представленного в книге.

Новосибирск,
Академгородок,
1983

А. П. Ершов

ПРЕДИСЛОВИЕ

В этой книге сделана попытка собрать и систематизировать разнообразные алгоритмы решения теоретико-графовых задач, возникающих в системном программировании. Большинство из них разбросано по многочисленным журнальным статьям, многие из которых не переводились на русский язык. Книга написана на основе курса лекций, который автор читал в течение ряда лет в Новосибирском государственном университете. Инициатива в разработке такого курса принадлежит академику А. П. Ершову.

Вышедшее в 1978 г. учебное пособие по этому курсу «Теория графов и программирование» привлекло внимание специалистов и выявило имеющийся интерес к данной тематике.

По мере работы над книгой выявлялись все новые и новые задачи и алгоритмы, но ограниченный объем книги заставил многое оставить за бортом. Так, в книгу не вошли такие вопросы, как вложения графов, теоретико-графовые методы конструирования баз данных и организации информационного поиска, графы адресуемых данных, алгоритмы анализа программ, основанные на итеративном алгоритме Килдала, графы в Р-технологии и т. д.

Книга написана как книга по теории графов, поэтому в главах 2—5 ничего не говорится о сферах применения того или иного алгоритма; этому посвящена глава 6. Все алгоритмы излагаются в некотором едином виде, часто называемом алголоподобным, с разной степенью детализации и без обсуждения реализации их на ЭВМ. Достаточно кратко даются обоснования алгоритмов, часто это просто формулировки необходимых утверждений.

Книга предназначена в первую очередь для специалистов в области системного программирования, но

может быть использована в качестве учебного пособия при организации соответствующего курса лекций для студентов специальности «Прикладная математика». При этом следует принять во внимание, что материал рассчитан на изучение в течение года при условии разбора если не всех, то большинства доказательств. Для ее чтения требуется подготовка в объеме первых двух курсов университета. Главы книги, за исключением первой, в достаточной мере независимы друг от друга в описании алгоритмов, но в остальном желательна последовательная проработка. Все отсылки к литературе делаются только в отношении доступных источников, в основном же все необходимые сведения приводятся.

В заключение хочу выразить благодарность С. С. Лаврову, А. П. Ершову, И. В. Поттосину и В. Н. Касьянову за постоянную поддержку и помощь в организации курса лекций и в написании книги. Особую благодарность я выражаю своей жене Людмиле Анатольевне Евстигнеевой, взявшей на себя тяжелый труд оформления всех вариантов рукописи книги.

В. А. Евстигнеев

ГЛАВА 1

ОСНОВНЫЕ ПОНЯТИЯ

§ 1. Основные определения теории графов

1.1. *Ориентированным графом* или, короче, *орграфом* $G = (X, \Gamma)$ называется упорядоченная пара (X, Γ) , где X есть непустое множество объектов некоторой природы — *вершин* графа, а Γ — многозначное отображение множества X на себя.

Поскольку многозначное отображение $\Gamma: X \rightarrow X$ полностью определяется перечислением пар вида (x, y) , $y \in \Gamma x$, то орграф можно определить и так.

Ориентированным графом $G = (X, U)$ называется упорядоченная пара (X, U) , где X есть непустое множество *вершин* орграфа, а U есть множество упорядоченных пар элементов из X , $U = \{(x, y)\} \subseteq X \times X$, называемых *дугами* орграфа.

Пусть u — дуга орграфа вида (x, y) , $u = (x, y)$. Вершина x называется *началом* дуги u , а вершина y — *концом* дуги u . При этом говорят, что дуга u *выходит* или *исходит* из вершины x и *входит* или *заходит* в вершину y . И в том, и в другом случае говорят, что дуга *инцидентна* соответствующей вершине. Дуга вида (x, x) называется *петлей*; вершины x и y , соединенные дугой, называются *смежными*.

Второе определение орграфа допускает обобщение на случай, когда пара вершин x и y может быть соединена несколькими параллельными дугами вида (x, y) . Полученная конструкция носит название *ориентированного мультиграфа*. От этого случая нужно отличать случай *симметричного* графа, у которого существование любой дуги вида (x, y) влечет существование дуги вида (y, x) . Ориентированный граф называется

полным, если каждая пара вершин в нем соединена дугой. Очевидно, что полный орграф симметричен.

Число дуг, заходящих в вершину x , называется *полустепенью захода* вершины x и обозначается $\deg^+(x)$. Число дуг, исходящих из вершины x , называется *полустепенью исхода* вершины x и обозначается $\deg^-(x)$. Общее число дуг, инцидентных вершине x , называется *степенью* вершины и обозначается $\deg x$. Имеем: $\deg x = \deg^+(x) + \deg^-(x)$.

Множество дуг, заходящих в вершину x , обозначается через $B(x)$; таким образом, $|B(x)| = \deg^+(x)$. Аналогично, через $A(x)$ обозначается множество дуг, исходящих из x ; таким образом, $|A(x)| = \deg^-(x)$. Множество вершин, являющихся началами дуг из $B(x)$, обозначается через $\Gamma^{-1}x$, а множество вершин, являющихся концами дуг из $A(x)$, обозначается через Γx . Вершины из множества $\Gamma^{-1}x$ называются *предшественниками* вершины x , а вершины из множества Γx — *потомками* вершины x . Если вершина x имеет только одного предшественника, то говорят о *предке* вершины x .

Вершина x , инцидентная только дугам вида (x, x) , называется *голой*. Вершина, не инцидентная ни одной дуге, называется *изолированной*. Степень изолированной вершины равна нулю. *Входом* или *началом* (начальной вершиной) орграфа называется вершина s , у которой $\deg^+(s) = 0$. *Выходом* или *конечной вершиной* графа называется вершина t , у которой $\deg^-(t) = 0$.

Иногда мы будем брать в качестве входа или выхода орграфа некоторые выделенные вершины. Это не ограничивает общности определения, так как всегда можно предполагать, что такое соглашение есть указание на появление фиктивного входа, из которого идет дуга в выделенную вершину, или на появление фиктивного выхода, в который заходит дуга из вершины, объявленной выходом. Введение фиктивного выхода позволяет нам не разделять случаи одного или нескольких выходов в орграфе.

Путь $\mu[a, b]$ из вершины a в вершину b называется последовательность вершин и дуг вида $a(a, x_1) \times x_1(x_1, x_2)x_2 \dots x_{n-1}(x_{n-1}, b)b$. Путь называется *простым*, если ни одна вершина в нем не встречается дважды. (В записи пути нет необходимости указывать и входящие в него дуги, и вершины.) Путь из входа s орграфа в выход t называется $s-t$ -*путем*. Будем считать, что хотя бы один такой путь в орграфе суще-

ствуется. Если в орграфе вершины a и b связаны путем $\mu[a, b]$, то говорят, что вершина b *достижима* из вершины a или что вершина a *достигает* вершину b . Орграф называется *односторонне связным*, если для любой пары вершин одна достижима из другой. Орграф называется *сильно связным*, если для любой пары вершин каждая из них достижима из другой. Путь называется *эйлеровым*, если он содержит все дуги графа ровно по одному разу, и *гамильтоновым*, если он содержит все вершины ровно по одному разу.

Путь, начало и конец которого совпадают, т. е. путь вида $\mu[a, a]$, называется *контуром* (с начальной вершиной a). Контур называется *простым*, если ни одна вершина в нем не повторяется дважды, *эйлеровым*, если он содержит все дуги графа в точности по одному разу, и *гамильтоновым*, если он содержит все вершины графа в точности по одному разу. *Длиной пути* или *контура* называется число дуг, входящих в него. Контур длины 1 есть *петля*. Путь длины 0 есть *тривиальный* или *вырожденный* путь. Длина эйлерова пути или контура равна числу дуг в орграфе; длина гамильтонова контура равна числу вершин в орграфе, длина гамильтонова пути на единицу меньше числа вершин.

Орграф $H = (\hat{X}, V)$ называется *частичным графом* графа $G = (X, U)$, если $\hat{X} \subseteq X$ и $V \subseteq U$, и *суграфом*, если $\hat{X} = X$ и $V \subseteq U$. Орграф $H = (Y, V)$ называется *подграфом* графа $G = (X, U)$, если $Y \subseteq X$ и из того, что $(y_1, y_2) \in U$ и $y_1, y_2 \in Y$, следует, что $(y_1, y_2) \in V$. Другими словами, частичный граф есть орграф, порождаемый некоторым подмножеством дуг исходного орграфа вместе с их концами. Суграф обязательно имеет то же множество вершин, что и исходный (т. е. если порождающее множество дуг инцидентно не всем вершинам исходного графа, то частичный граф дополняется до суграфа за счет изолированных вершин). Подграф порождается некоторым подмножеством вершин исходного графа и теми дугами, оба конца которых принадлежат указанному подмножеству.

Сильно связный подграф орграфа называется *зоной*. Зона, максимальная относительно включения вершин, называется *компонентой сильной связности* или *бикомпонентой*.

Вершина p_0 есть *начальная вершина* подграфа H орграфа G с входом s , если либо p_0 совпадает с s , либо

вершина p_0 имеет предшественника в G , не принадлежащего H . Вершина q_0 называется *входной вершиной* подграфа H , если существует путь из s в q_0 , не содержащий вершин подграфа, отличных от q_0 . Внешний потомок вершины подграфа H называется его *выходной вершиной*.

Подграф H графа G , имеющий одну начальную и одну выходную вершины, называется *гаммом*, если выходная вершина не является предшественником начальной. На рис. 1.1 показаны гаммы орграфа G . Гаммаи G_2 , G_3 и G_4 одновременно являются бикомпонентами, а подграф, порожденный вершинами v_1 и v_3 , является зоной.

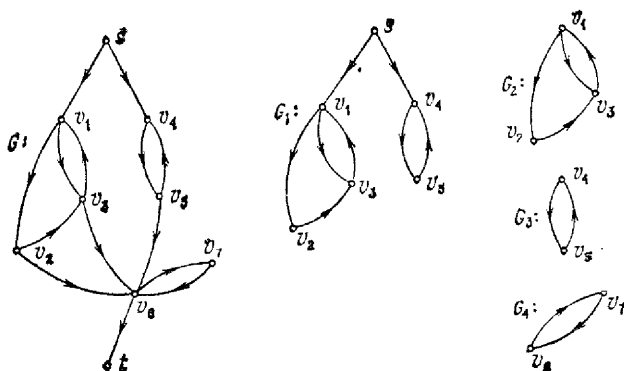


Рис. 1.1.

Линейной компонентой орграфа G называется гамма, удовлетворяющий условиям:

- а) начальная и выходная вершины гамма лежат на каждом $s-t$ -пути орграфа G ;
- б) начальная вершина гамма не достижима из выходной;
- в) не существует другого гамма, который удовлетворял бы первым двум условиям и содержался бы в данном гамме.

Гаммаи G_1 и G_2 на рис. 1.1 являются линейными компонентами орграфа G .

1.2. Неориентированным графом или просто графом $G = (X, U)$ называется упорядоченная пара (X, U) , где X есть множество вершин графа, а U есть множество неупорядоченных пар элементов из X , называемых ребрами графа. Из контекста всегда ясно, о каком типе

графа — ориентированном или неориентированном — идет речь.

В общем случае у ребер не различают начало и конец, и запись (x, y) обозначает то же ребро, что и (y, x) . Иногда при обходах графа или при движении по ребрам из одной вершины в другую мы будем говорить о начале ребра и его конце, подразумевая при этом, что направление движения создает ориентацию ребра. В графе каждая пара вершин соединена не более чем одним ребром; допущение параллельных ребер приводит к новой конструкции, именуемой *мультиграфом*.

На случай неориентированных графов переносятся почти все введенные выше понятия. Так, две вершины *смежны*, если они соединены ребром; ребро *инцидентно* своим концам, *степень* вершины есть число инцидентных ей ребер (полустепени захода и исхода теряют свой смысл). Говоря о входе или выходе в неориентированном графе, будем иметь в виду некоторые выделенные вершины; например, откуда начинается и где заканчивается обход.

Для пути и контура в неориентированном графе имеются аналоги: цепь и цикл. *Цепью* $\mu[a, b]$, соединяющей вершины a и b , называется последовательность вершин и ребер вида $a(a, x_1)x_1(x_1, x_2)x_2 \dots x_{n-1}(x_{n-1}, b)b$. Цепь, начало и конец которой совпадают, называется *циклом*. Термин «цепь» иногда заменяется термином «путь». Употребление термина «цикл» в тех случаях, когда речь идет об ориентированных графах, не ведет к путанице, так как оно ограничено узкими областями (например, потоки по сетям). Для цепей и циклов неориентированного графа аналогично тому, как это делалось для орграфов, определяются понятия простоты, эйлеровости, гамильтоновости, а также длины. Понятия односторонней и сильной связности теряют смысл, их аналогом служит понятие связности: граф называется *связным*, если каждая пара вершин связана цепью. Сохраняются понятия частичного графа, суграфа и подграфа с естественной заменой множества дуг на множество ребер.

1.3. Деревом называется связный неориентированный граф без циклов. Это не единственное возможное определение. Теорема 1 устанавливает эквивалентность различных свойств дерева, каждое из которых может служить определением.

Теорема 1. Для графа $G = (X, U)$ следующие утверждения эквивалентны:

- 1) $G \sim$ дерево;
- 2) любые две вершины в графе G соединены единственной простой цепью;
- 3) граф G связен и имеет $|X| - 1$ ребер;
- 4) граф G не содержит циклов и имеет $|X| - 1$ ребер;
- 5) граф G не содержит циклов, но добавление ребра между любыми двумя несмежными вершинами приводит к появлению одного цикла;
- 6) граф G связен, но утрачивает это свойство после удаления любого ребра.

Мы опускаем доказательство этой теоремы. Пример дерева приведен на рис. 1.2, а.

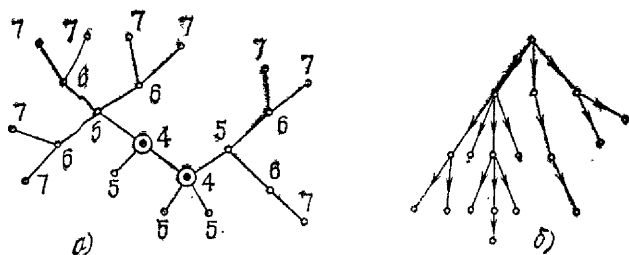


Рис. 1.2.

Вершина степени один называется *висячей*. Из определения дерева и теоремы 1 вытекает, что всякое дерево имеет по крайней мере две висячие вершины.

Определим расстояние $d(x, y)$ между вершинами x и y в дереве как длину пути из x в y . Расстояние от вершины x до наиболее удаленной от нее вершины называется *эксцентриситетом* $e(x)$ вершины x , т. е. $e(x) = \max_y d(x, y)$. Наибольший из эксцентриситетов называется *радиусом* $r(T)$ дерева T . *Центральной вершиной* дерева T называется вершина, у которой эксцентриситет равен радиусу. *Центром* дерева называется множество его центральных вершин. Для дерева на рис. 1.2, а возле каждой вершины указан ее эксцентриситет. Центр этого дерева состоит из двух вершин с эксцентриситетом 4. Можно показать, что каждое дерево имеет центр, состоящий или из одной вершины, или из двух смежных вершин.

Назовем ветвью к вершине x дерева T максимальное поддереву, содержащее x в качестве висячей вершины. Таким образом, число ветвей к вершине x равно ее степени. Вес вершины x есть наибольшее число ребер в ветви к вершине x . Вес каждой висячей вершины в дереве равен числу ребер в этом дереве. Вершина x называется *центроидной*, если вершина x имеет наименьший вес. *Центроидом* дерева называется множество его центроидных вершин. Можно показать также, что каждое дерево имеет центроид, состоящий из одной вершины или из двух смежных вершин.

1.4. *Ориентированным корневым деревом* $T(r)$ или *ордеревом с корнем r* называется орграф с выделенной вершиной r , который удовлетворяет следующим условиям:

а) $T(r)$ — дерево, если не принимать во внимание ориентацию дуг;

б) из корня r достижима любая вершина (или корень r достижим из любой вершины);

в) в корень r не заходит ни одна дуга (или из корня не выходит ни одна дуга).

Ясно, что любое дерево можно превратить в ордереву, если выбрать некоторую вершину в качестве корня и задать ориентацию дуг. Для определенности будем считать, что все дуги ориентированы от корня. Пример ордерева показан на рис. 1.2, б.

Плоским топологическим графом называется изображение графа на плоскости, при котором разным вершинам графа отвечают разные точки плоскости, ребрам — дуги кривых (без самопересечений) и никакие два ребра не имеют общих точек, отличных от их концов. Плоские топологические графы, которые можно преобразовать друг в друга непрерывной деформацией плоскости, не считаются различными.

Плоский топологический граф G называется *упорядоченным деревом*, если:

а) G — ордереву с корнем;

б) для каждой вершины (включая корень) задан относительный порядок максимальных поддеревьев, имеющих данную вершину своим корнем.

Легко видеть, что если два дерева, отличающихся друг от друга только относительным порядком поддеревьев вершин, не считать различными, то мы приходим к понятию ордерева, поскольку в нем имеет значение только относительная ориентация вершин,

а не их порядок. На рис. 1.3 изображены два различных упорядоченных дерева.

Важным типом упорядоченных деревьев являются так называемые *бинарные деревья*, определяемые рекурсивно следующим образом:

а) одновершинное дерево есть бинарное дерево;

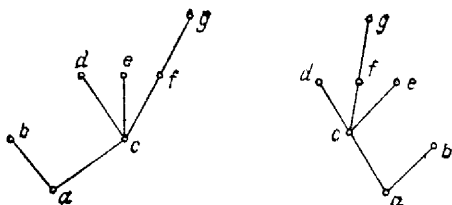


Рис. 1.3.

б) тройка (T_l, v_0, T_r) есть бинарное дерево с корнем v_0 , левым поддеревом T_l и правым поддеревом T_r , причем T_l и T_r — бинарные деревья (возможно, пустые); см. рис. 1.4, а.

Бинарное дерево не является частным случаем упорядоченного, но эти понятия тесно связаны. Например, бинарные деревья на рис. 1.4, б и 1.4, в различны между собой (в первом корень имеет пустое левое поддерево, во втором — пустое правое), но как упорядоченные деревья они идентичны (изоморфны).

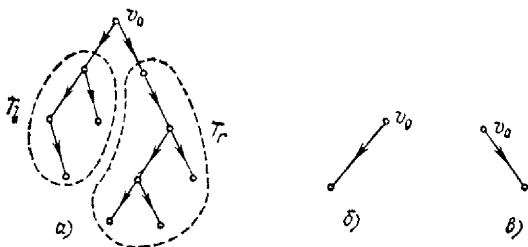


Рис. 1.4.

Обобщением бинарных деревьев являются *t-арные деревья*, в которых каждая вершина является корнем не более чем t поддеревьев, каждое из которых в свою очередь есть t -арное дерево (возможно, пустое). *Каркасом* или *остовом* неориентированного графа называется его суграф в виде дерева. Граф G имеет каркас

тогда и только тогда, когда он связан. В самом деле, если граф связан, то выясним, имеется ли в нем ребро, удаление которого не нарушает связности. В случае, когда таких ребер нет, граф G сам является деревом в силу свойства 6) теоремы 1; если же такое ребро есть, то удалим его и выясним, имеется ли в полученном графе ребро, удаление которого не нарушает связности, и т. д. Когда удаление ребер без нарушения связности станет невозможным, получим дерево, множество вершин которого совпадает с множеством вершин G . Проведенное рассуждение дает алгоритм для отыскания некоторого каркаса графа.

Для орграфа можно определить понятие *оркаркаса*, т. е. суграфа в виде корневого ордерова. В орграфе G существует оркаркас с корнем в вершине r , если все вершины графа достижимы из r .

1.5. Другие типы графов. Граф G называется *несепарабельным* или *двусвязным*, если удаление любой его вершины не нарушает связности. Максимальная относительно включения вершин компонента сепарабельности (компонента двусвязности) называется *блоком*. В любом блоке существует простой цикл, проходящий через два произвольным образом выбранных ребра.

Вершина, общая двум или более блокам, называется *точкой сочленения* или *разделяющей вершиной*. Так как удаление точки сочленения нарушает связность графа, то наличие в графе таких вершин наводит на него некоторую деревообразную структуру. В самом деле, замена каждого блока ребром с концами в разделяющих вершинах или просто висячим ребром, если соответствующий блок имеет только одну точку сочленения, превращает граф в дерево. Для описания такой деревообразной структуры используется понятие *графа блоков* $F(G)$ графа G : это граф, вершины которого соответствуют блокам B и разделяющим вершинам z графа G (рис. 1.5), причем вершины B и z соединены ребром при $z \in B$, а вершины B между собой не смежны. Аналогом блока в ориентированных графах служит линейная компонента.

Полным n -вершинным графом K_n называется неориентированный граф, все вершины которого смежны между собой. Для многих приложений важным является выявление наличия в графе полных подграфов, или *клик*, достаточно большого размера, ибо к этой

задаче сводится задача об отыскании максимального множества попарно несмежных вершин.

Двудольным графом $G = (X_1, X_2, U)$ называется граф, вершины которого разбиты на два непересекающихся подмножества X_1 и X_2 так, что концы ребер

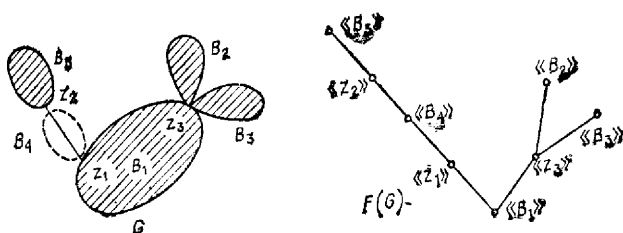


Рис. 1.5.

$(x, y) \in U$ принадлежат разным подмножествам, т. е. $x \in X_1$ и $y \in X_2$. Важное свойство двудольных графов: все циклы имеют четную длину. Заметим, что здесь нуль считается четным числом, так что к двудольным графам относятся и деревья.

Важной особенностью графов, позволяющей использовать их для моделирования систем и процессов, является возможность приписывать ребрам (дугам) и вершинам *вес*, в качестве которых чаще всего выступают числа или наборы чисел. Без ограничения общности можно считать, что во взвешенном графе вес приписаны только ребрам (дугам). Исключения будут оговариваться особо. Во взвешенном графе длина пути (контура, цепи, цикла) определяется как сумма или произведение весов входящих в него дуг. Существуют более сложные способы определения длины пути (во всех таких случаях это оговаривается особо). Веса дуг часто называют *длинами*, *стоимостью*, *временем* и т. д. в зависимости от конкретной ситуации.

1.6. Матрицей смежности $A(G)$ графа G с n вершинами называется квадратная матрица порядка n с элементами a_{ij} : $a_{ij} = 1$, если существует дуга (ребро) (x_i, x_j) ; $a_{ij} = 0$ в противном случае. Для неориентированного графа матрица смежности симметрична относительно главной диагонали. В случае ориентированного графа входу соответствует столбец из нулей, выходу — строка из нулей, число единиц в i -й строке равно полустепенни исхода вершины x_i , число единиц в j -м столбце равно полустепенни захода вершины x_j .

Матрица смежности полностью определяет структуру графа (с точностью до изображения на плоскости и нумераций вершин), в том числе и взвешенного (в матрице смежности которого элемент a_{ij} полагаем равным весу дуги (x_i, x_j)). Пусть $G_1 = (X, U_1)$, $G_2 = (X, U_2)$ — два графа с одним и тем же множеством вершин, A_1 и A_2 — их матрицы смежности. Тогда матрица $A_1 \oplus A_2$, где символом \oplus обозначено логическое сложение, определяет объединение графов, т. е. граф $G_1 \cup G_2 = (X, U_1 \cup U_2)$. Предположение о равенстве множеств вершин в графах — не ограничение, так как множество вершин графа всегда можно пополнить изолированными вершинами, соответствующими отсутствующим в графе G_1 вершинам из графа G_2 , и наоборот. При этом матрицы смежности будут иметь один и тот же размер и i -й вершине одного графа будет соответствовать i -я вершина другого.

Матрицей инцидентности $B(G)$ графа G с n вершинами и m дугами называется прямоугольная матрица размера $n \times m$ с элементами b_{ij} : $b_{ij} = 1$, если вершина x_i есть начало дуги u_j ; $b_{ij} = -1$, если вершина x_i есть конец дуги u_j ; $b_{ij} = 0$ в противном случае. Заметим, что в определении отсутствует случай, когда в вершине x_i имеется петля. Большинство авторов использует дополнительное значение для b_{ii} (например, какой-нибудь символ или выражение «1 и -1»).

Матрицей достижимости $R(G)$ графа G с n вершинами называется квадратная матрица порядка n с элементами r_{ij} : $r_{ij} = 1$, если вершина x_j достижима из x_i ; $r_{ij} = 0$ в противном случае.

§ 2. Графы как модели программ, данных и процессов

2.1. Управляющие графы. Многие задачи анализа программ, возникающие при оптимизации, трансляции, проверке правильности, тестировании и т. д., значительно упрощаются, если их рассматривать на теоретико-графовых моделях. В основе моделей лежит управляющий граф программы. Орграф $G = (X, U)$ называется *управляющим графом* (*p-графом*, *графом переходов*), если выполнены следующие условия:

- а) граф G не содержит параллельных дуг;
- б) в множестве вершин графа выделена одна вершина s — вход графа;

в) в множестве вершин графа выделена хотя бы одна вершина t — выход графа;

г) каждая вершина $x \in X$ достижима из входа s ;

д) каждая вершина $x \in X$ достигает выхода t .

З а м е ч а н и е. Можно считать, что в управляющем графе имеется более чем один выход, поскольку этот случай легко сводится к случаю одного выхода введением фиктивного выхода, соединенного дугами с каждым фактическим выходом. В большинстве случаев можно считать, что полустепень исхода каждой вершины, отличной от входа и выхода, равна 1 или 2, а полустепень исхода входа равна 1.

Построение управляющего графа для конкретной программы проводится по простым правилам с требуемой степенью детальности (и, как правило, автоматически). Тривиальное решение задачи состоит в изображении каждого оператора языка (машинной команды, строки или любого фрагмента, признаваемого единицей языка) отдельной вершиной. При этом две вершины смежны, если между соответствующими операторами есть передача управления. Точнее, оператор, после исполнения которого производится передача управления, представляется началом дуги; оператор, на который передается управление, — концом дуги, а каждая передача управления — дугой. При таком подходе размер графа сильно растет за счет появления длинных цепочек вершин, соответствующих линейным участкам программ. Они могут быть представлены одной вершиной. Более сложные подходы к построению управляющих графов состоят в выделении различного рода квазилинейных участков, как это было сделано, например, при разработке транслятора с языка альфа. Это возможно, если нет необходимости тщательно учитывать внутреннюю структуру отдельных частей программы.

Для примера рассмотрим построение управляющего графа для программы BUBBLE, написанной на алголоподобном языке. Текст программы приведен на рис. 1.6, соответствующий управляющий граф — на рис. 1.7. В этом графе каждая вершина соответствует одному оператору, для сравнения на рис. 1.8 приведен редуцированный управляющий граф, в котором каждый линейный участок представлен одной вершиной. В качестве еще одного примера приведен управляющий граф на рис. 1.10 для фрагмента программы, написанной на языке эльбрус эль-76 (рис. 1.9).

```

1. SUBROUTINE BUBBLE(A, N)
2. BEGIN
3. FOR I=2 STEP 1 UNTIL N DO
4. BEGIN
5. IF A(I) GE A(I-1) THEN GOTO NEXT
6. J = 1
7. LOOP: IF J LE 1 THEN GOTO NEXT
8. IF A(J) GE A(J-1) THEN GOTO NEXT
9. TEMP = A(J)
10. A(J) = A(J-1)
11. A(J-1) = TEMP
12. J = J - 1
13. GOTO LOOP
14. NEXT: NULL
15. END
16. END

```

Рис. 1.6.

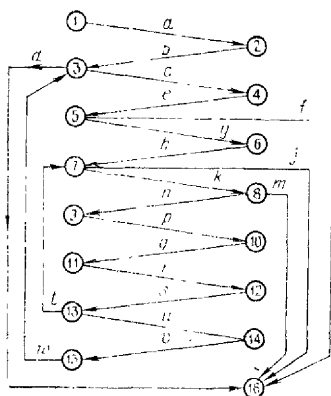


Рис. 1.7.

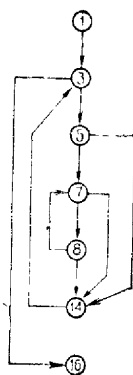


Рис. 1.8.

```

1. если  $k=N$  то
2. для  $i$  от 1 до  $N$ 
3. цикл
4.  $mas[i, 1] := c1$ ;
5.  $mas[i, 2] := c1 \times c1$ ;
6.  $mas[i, 3] := c1 \times mas[i, 3]$ ;
7. повторить;
8.  $k := N + 1$ 
9. иначе  $k := 0$ 
10. все;

```

Рис. 1.9.

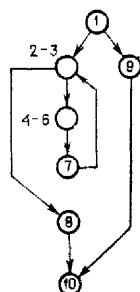


Рис. 1.10.

2.2. Марковская граф-модель программы представляет собой взвешенный управляющий граф программы, причем веса дуг (а также вершин) являются фиксированными значениями некоторых параметров. Так, вершинам обычно сопоставляется объем и/или время вычислений (время передачи управления полагают равным нулю). Каждой дуге, соответствующей передаче управления от i -го к j -му оператору, приписывается постоянное значение q_{ij} вероятности осуществления передачи (события выбора переходов независимы). Для правильно построенной программы и ее граф-модели можно из входа попасть последовательно по дугам в любую вершину, а из любой вершины — в выход. Какая-либо реализация счета задается указанием $s-t$ -пути (не обязательно простого). Вероятность реализации определяется как произведение соответствующих вероятностей q_{ij} . Ограничений на конфигурацию программы и множество реализаций счета граф-модель не накладывает. Множество реализаций счета есть подмножество всех $s-t$ -путей. Для завершения построения модели нужно соединить дугой конечную (выход) и начальную (вход) вершины граф-модели, так что вход становится концом этой дуги.

2.3. Полумарковская модель. Иногда целесообразно представлять время исполнения операторов случайной величиной (например, если соответствующая вершина есть объединение нескольких операторов, прерываемых условными переходами). Граф-модель в этом случае будет полумарковской, причем время исполнения операторов будет характеризоваться средним значением μ и дисперсией σ . Случайные величины, определяющие время исполнения, считаются взаимно независимыми. Если все дисперсии σ равны нулю, то модель превращается в марковскую.

Основная трудность состоит в определении вероятностей передачи управления. Последние могут быть определены либо заранее, если ветвление происходит после операций с индексами, либо по известным распределениям входных данных, если ветвление происходит после анализа входных данных, либо путем аппроксимации или моделирования, если распределение вероятностей для входных данных неизвестно. Расчет вероятностей осуществим, когда они являются функциями параметров решаемой задачи. Иногда обнаруживается связь между количеством выполнений операторо-

ров, числом циклов в программе и вероятностями переходов.

2.4. Пусть A — множество операторов некоторой программы, B^+ — множество входов, B^- — множество выходов всех операторов, причем $B^+ \cap B^- = \emptyset$ и $B = B^+ \cup B^-$. Информационным графом $I = (B, U)$ называется орграф без петель, вершинами которого являются входы и выходы операторов, а дуга (x, y) принадлежит множеству дуг U тогда и только тогда, когда $x \in B^-$ и $y \in B^+$. Дуги информационного графа называют информационными парами, сохраняя традиционное название «дуга» для дуг управляющего графа.

Операторной схемой над общей памятью или O -схемой называется упорядоченная четверка (A, G, \hat{X}, R) , где A — непустое множество операторов, \hat{X} — конечное множество неопределяемых объектов, называемых величинами, G — управляющий граф, $R: B \rightarrow \hat{X}$ — отображение множества входов и выходов операторов на множество величин.

Операторной схемой над распределенной памятью или P -схемой называется упорядоченная тройка (A, G, I) , где A, G определяются, как и выше, а I — информационный граф.

Пример информационного графа и операторных схем показан на рис. 1.11 (а) управляющий граф G ; б) информационный граф I ; в) P -схема).

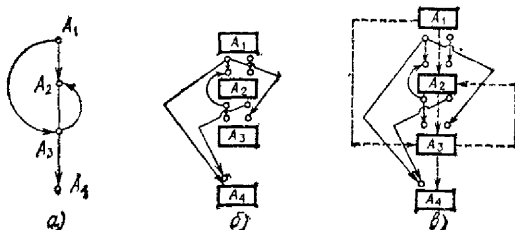


Рис. 1.11.

2.5. Информационные связи в операторной схеме группируются в *связки*. В одну связку входят все информационные связи, выходящие из некоторого оператора (результата). Для правильного распределения памяти нужно, чтобы всем аргументам (величинам, сопоставленным входам операторов) и результату (величине, сопоставленной выходу оператора), входящим

в одну и ту же связку, была сопоставлена одна и та же величина. Экономия памяти достигается тогда, когда нескольким связкам удастся сопоставить одну и ту же величину без нарушения информационных связей. Для этого систему связок информационного графа изображают в виде неориентированного графа — *графа несовместимостей*, у которого вершины соответствуют информационным связкам, а две вершины смежны тогда и только тогда, когда связки несовместимы.

2.6. Ордерево как форма организации динамических словарей. Пусть на вход некоторой программы подается некоторый текст и требуется составить список всех слов (различных), встречающихся в тексте. Эта задача может быть усложнена требованием предоставления возможности пользоваться списком до включения в него всех слов. Кроме того, неизвестно общее число слов в тексте. Для решения задачи с достаточной эффективностью требуется структура, которую можно было бы просмотреть за время, меньшее, чем потребовалось бы для неупорядоченного списка слов, и которая обладала бы еще одним свойством: вписывание новых слов не должно вызывать затруднений. Такой структурой служит корневое упорядоченное дерево в виде бинарного (или *m*-арного) *дерева сортировки* — бинарного дерева, снабженного метками, взятыми из множества меток с введенным на нем отношением полной упорядоченности, и такого, что все метки из левого поддерева любой вершины меньше метки в этой вершине, а метки в правом поддерева — соответственно больше.

Включение нового слова соответствует подвешиванию к дереву новой вершины, меткой которой было бы новое слово, без нарушения свойства дерева быть бинарным. Для отыскания места подвешивания новой вершины будем двигаться, начиная с корня и сравнивая включаемое слово с метками проходимых вершин: если метка совпадает с включаемым словом, то ничего делать не надо, так как слово уже в словаре; если метка больше нового слова, то движемся к потомку вершины по левой дуге, в противном случае — по правой. Если слова в словаре нет, то мы попадаем в вершину, у которой отсутствует один или оба потомка; к ней мы и подвешиваем новую вершину. Можно представить себе ситуацию, когда дерево сортировки вырождается в линейный список; тогда для сохранения структуры де-

рева с нужными свойствами применяются специальные приемы, рассматриваемые в главе 4.

2.7. Граф-модель вычислительного процесса. Пусть дана некоторая формула, или задача, состоящая из ряда подзадач, или совокупность задач, использующих одни и те же ресурсы, и т. п. Для простоты будем говорить о вычислении некоторой формулы. Порядок вычислений может существенно влиять на объем требуемой для хранения исходных и промежуточных данных оперативной памяти. Изобразим формулу в виде *каскадного информационного графа*. В нем каждая внутренняя вершина соответствует оператору; это означает, что в эту вершину входит столько дуг, сколько аргументов у оператора, а выходит одна дуга, представляющая результат вычислений. Вводимым данным будут соответствовать вершины, из которых исходит только одна дуга и ни одна дуга не входит, — входы графа. Отсюда следует, что каскадный информационный граф представляет собой корневое ордеререво, в котором корень соответствует вычисляемому результату. В качестве веса вершины в такой модели вычислительного процесса выступает требуемый объем памяти.

2.8. Граф-модель информационно-логической системы. Среди исходных данных, необходимых для выбора структуры комплекса вычислительных средств (КВС), важнейшую роль играют сведения о множестве задач, решаемых системой. Простейшей формой задания такого множества является перечень задач; следующей по сложности формой является комплект алгоритмов решения задач системы. Наиболее развитой и совершенной формой задания вычислительной работы системы является *информационно-логическая структура* (ИЛС) системы. Под ИЛС некоторой системы управления понимается совокупность алгоритмов, увязанных в единое целое по перерабатываемой информации, для решения общей главной задачи системы.

Описание ИЛС системы должно включать в себя не только перечень алгоритмов, но и указания о последовательности их реализации, об их характеристиках и об исходной информации. Наиболее удобная, наглядная и многосторонняя форма описания ИЛС — представление ее в виде орграфа алгоритмов. Такие графы позволяют не только выявлять свойства ИЛС, но и решать следующие важные практические задачи:

— установление рационального порядка разработки алгоритмов;

— перечисление возможных вариантов привязки алгоритмов к пунктам обработки информации;

— составление исходных матриц, необходимых для оптимизации КВС по показателям качества КВС;

— упрощение ИЛС системы;

— составление управляющих (диспетчерских) алгоритмов.

Графом алгоритмов системы называется оргграф $G = (X, U)$, вершины x_i которого изображают частные реализации i -х алгоритмов системы. Две вершины x_i и x_j соединяются дугой (x_i, x_j) , если алгоритм A_j может реализоваться только вслед за алгоритмом A_i . Вершинам графа алгоритмов приписывается вес в форме некоторых физических величин, связанных с реализацией этих алгоритмов, например числа операций N_i , требуемых для реализации i -го алгоритма, времени t_i реализации алгоритма на ЭВМ, ошибок определения тех или иных величин в результате реализации i -го алгоритма и т. д.

В ИЛС системы управления соотношения следования между алгоритмами обычно соответствуют направлениям потоков информации, которые циркулируют в системе при реализации алгоритмов. В самом деле, соотношение $x_i < x_j$ (алгоритм A_j следует за A_i) однозначно утверждает, что для реализации A_j необходима некоторая входная информация, которая получается в результате реализации алгоритма A_i . Вследствие этого в графах алгоритмов не только вершинам, но и дугам могут быть приписаны вполне определенные веса. Это не что иное, как направленные потоки информации между последовательными алгоритмами, описываемые ее составом, объемом и точностными характеристиками. В некоторых случаях дугам графа может быть приписано другое физическое содержание, например вероятность p_{ij} перехода от вершины x_i к вершине x_j .

§ 3. Графы как объекты обработки информации

3.1. Представление графов в памяти ЭВМ. Представление графов в памяти ЭВМ существенно зависит от типов структур данных, допускаемых используемым алгоритмическим языком и типом ЭВМ.

Представление с помощью матрицы смежности — одно из самых распространенных, ведущих свое начало от удобства ручной работы с матрицей смежности и удобства описания алгоритмов на графах, заданных в таком виде. Для графов с большим числом дуг это достаточно компактное представление, особенно если есть возможность работать с двоичными битами в машинном слове, как это допускает, например, язык ПЛ/1. К недостаткам следует отнести большой расход памяти при работе с графами, имеющими небольшое число дуг (матрица смежности при этом получается весьма разреженной), а также невозможность уменьшения трудоемкости алгоритмов в том случае, когда она пропорциональна числу дуг, а не числу вершин.

Матрица смежности неориентированного графа симметрична относительно главной диагонали, поэтому достаточно хранить в памяти только половину ее. Задание графа с помощью матрицы смежности удобно еще и тогда, когда граф взвешенный и элементами матрицы являются не нули и единицы, а веса дуг. Пример задания ориентированного и неориентированного графов матрицами смежности приведен на рис. 1.12.

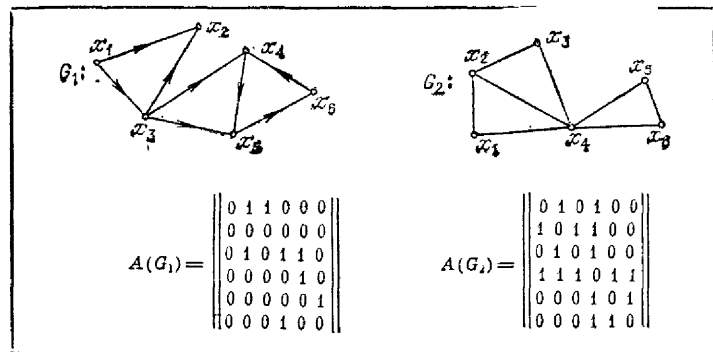


Рис. 1.12.

Представление с помощью матрицы инцидентий определяет граф однозначно (с точностью до изоморфизма), но применяется крайне редко в силу большой разреженности матрицы и практического отсутствия алгоритмов, работающих на такой структуре данных.

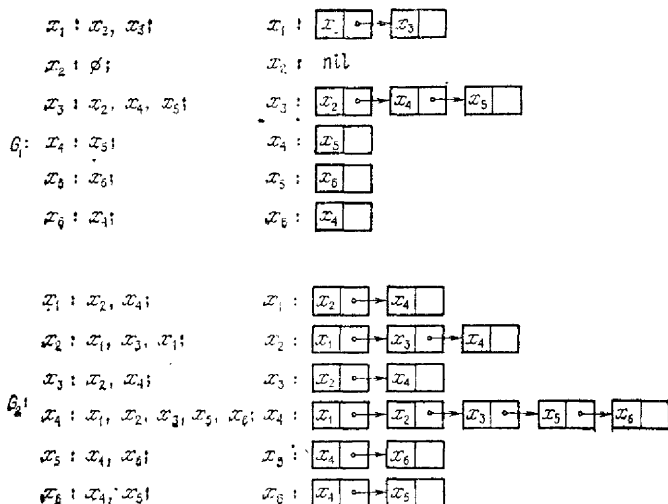
Представление с помощью списков смежности является главной альтернативой представлению с помощью матрицы смежности. Список смежности для вершины v

есть просто список вершин из множества Γ_v , т. е. это список концов дуг, исходящих из вершины v в случае орграфа, или список смежных с v вершин в случае графа. Граф представляется с помощью $|X|$ списков смежности, по одному для каждой вершины. Если число дуг в орграфе существенно мало по сравнению с полным графом, то этот способ представления весьма эффективен. Списки смежности занимают объем памяти $|X| + |E|$ и легко реализуются с помощью списочных структур. Часто используются различные варианты списков смежности, например, задаются списки вершин из $\Gamma^{-1}v$, совместные списки вершин из Γv и $\Gamma^{-1}v$ и т. д. Менее удобен этот способ представления для задания взвешенных графов, ибо тогда возникает дополнительная задача хранить где-то веса дуг и устанавливать соответствующие связи между дугами и их весами. Пример задания графов, показанных на рис. 1.12, с помощью списков смежности приведен на рис. 1.13.

Представление с помощью списка дуг применяется в тех случаях, когда необходимо иметь отдельную, независимую нумерацию дуг. При этом способе каждой дуге сопоставляется тройка $\langle u, x, y \rangle$, где u — дуга, x — ее начало, y — ее конец. Этот способ представления легко обобщается на случай взвешенных графов. Более того, он наиболее приспособлен для хранения различной информации о дугах.

Пусть G — неориентированный граф, $A(G)$ — его матрица смежности. Так как $A(G)$ симметрична относительно главной диагонали, будем рассматривать только ее верхнюю треугольную часть A' . Запишем строки A' последовательно одну за другой и рассмотрим полученную последовательность из нулей и единиц как двоичное число. Меняя нумерацию вершин, будем для одного и того же графа получать разные числа. Наибольшее из них определяется для графа однозначно и носит название *кода Харари*. Код Харари определяет граф однозначно, поэтому, например, задачу определения изоморфизма двух графов можно свести к сравнению соответствующих кодов Харари. Правда, этот метод столь же неэффективен, как и другие методы установления изоморфизма двух произвольных графов. Нумерация вершин (и матрица смежности), соответствующая коду Харари, носит название *канонической* и используется при перечислении (генерировании) графов с заданными свойствами.

Код Прюфера может быть использован для задания деревьев, поскольку допускает реализацию ограниченного числа алгоритмов, работающих с такой структурой данных. Пусть T — дерево с множеством вершин



$$G_1 = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \begin{vmatrix} 0 & x_2 & x_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_2 & 0 & x_3 & x_5 & 0 \\ 0 & 0 & 0 & 0 & x_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & x_6 \\ 0 & 0 & 0 & x_4 & 0 & 0 \end{vmatrix} \end{matrix}, \quad G_2 = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \begin{vmatrix} 0 & x_2 & 0 & x_4 & 0 & 0 \\ x_1 & 0 & x_3 & x_4 & 0 & 0 \\ 0 & x_2 & 0 & x_4 & 0 & 0 \\ x_1 & x_2 & x_3 & 0 & x_5 & x_6 \\ 0 & 0 & 0 & x_4 & 0 & x_6 \\ 0 & 0 & 0 & x_4 & x_5 & 0 \end{vmatrix} \end{matrix}$$

Рис. 1.13.

$\{v_1, \dots, v_n\}$. Будем считать, что номер вершины v_i равен i . Сопоставим дереву T последовательность $\{a_1, \dots, a_{n-2}\}$ по следующему правилу:

1. Полагаем i равным 1.
2. В последовательности

$$1, 2, \dots, n \quad (*)$$

путем просмотра слева направо ищем номер первой висячей вершины. Пусть это будет b_i .

3. Ищем, с какой вершиной смежна вершина b_i . Пусть это будет вершина a_i . Запоминаем a_i .
4. В последовательности (*) вычеркиваем b_i .
5. Из дерева T удаляем вершину b_i .
6. Полагаем $i := i + 1$.
7. Если $i < n - 1$, то переходим к шагу 2. Если $i = n - 1$, то выдаем последовательность $[a_1, \dots, a_{n-2}]$. Это и есть код Прюфера.

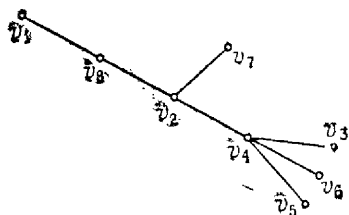


Рис. 1.14.

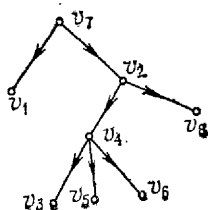


Рис. 1.15.

Например, для дерева на рис. 1.14 код Прюфера равен $[8, 4, 4, 4, 2, 2]$. Код Прюфера взаимно однозначно кодирует деревья. В случае корневого ордерера процедура получения кода Прюфера аналогична. Необходимо только на последнем месте указывать корневую вершину и при распаковке кода исключать эту вершину из просмотра последовательности (*). Так, для дерева на рис. 1.15 код Прюфера равен $[7, 4, 4, 4, 2, 2, 7]$.

Код Прюфера дает возможность доказать теорему Кэли о числе помеченных деревьев.

Теорема 2 (Кэли). Число помеченных n -вершинных деревьев равно n^{n-2} .

Код Прюфера является оптимальным по памяти кодированием деревьев. В самом деле, пусть $\mathcal{A} = \bigcup_{n=1}^{\infty} \mathcal{A}_n$,

где \mathcal{A}_n — конечное множество. Предположим, что при каком-то способе кодирования элементов из \mathcal{A} для запоминания одного элемента из \mathcal{A}_n используется самое большее $j(n)$ битов памяти. Кодирование $j(n)$ назовем оптимальным, если $\lim_{n \rightarrow \infty} \frac{j(n)}{H(n)} = 1$, где $H(n) = \log |\mathcal{A}_n|$, $|\mathcal{A}_n|$ — мощность множества \mathcal{A}_n . Для оценки энтропии $H(n)$ для множества деревьев с n вершинами по теореме 2 имеем $H(n) = (n-2) \log n$. Отсюда следует, что кодирование деревьев кодом Прюфера является оптимальным, так как для хранения кода необходимо $j(n) =$

$= (n - 2) \log n$ битов памяти, и поэтому $\lim_{n \rightarrow \infty} \frac{I(n)}{H(n)} = 1$.

При задании дерева списками смежности имеем $\lim_{n \rightarrow \infty} \frac{I(n)}{H(n)} = 3$ для неориентированных графов и $\lim_{n \rightarrow \infty} \frac{I(n)}{H(n)} = 2$ для ориентированных.

3.2. Для решения одной и той же задачи могут существовать различные по эффективности алгоритмы. Понятие эффективности алгоритма расплывчато, но чаще под этим понимается эффективность в смысле затрат времени и памяти ЭВМ. При этом с задачей связывается некоторое число (или совокупность чисел), называемое *размерностью* задачи, которое выражает меру количества входных данных. В задачах на графах таким параметром может служить число вершин (чаще всего), число дуг, пара чисел вида число вершин — число дуг и др.

Время, затрачиваемое алгоритмом, как функция размерности задачи называется *временной сложностью* или *трудоемкостью* алгоритма. Предельное значение сложности при неограниченном увеличении размерности называется *асимптотической временной сложностью* (*трудоемкостью*). Размер памяти, минимально необходимый для решения задачи алгоритмом, называется *емкостной сложностью* или *сложностью по объему памяти*. Аналогично асимптотической временной сложности можно определить *асимптотическую емкостную сложность*.

Если алгоритм обрабатывает входы размерности n , т. е. может решать задачи размерности n , за время cn^2 , где c — некоторая постоянная, определяемая реализацией алгоритма, то говорят, что временная сложность этого алгоритма есть $O(n^2)$. Несмотря на то, что при таком определении сложности алгоритма основное внимание уделяется порядку роста величин, нужно понимать, что алгоритм с большим порядком роста сложности может иметь меньшую постоянную, нежели алгоритм с меньшим порядком роста сложности. Поэтому его преимущества могут сказываться лишь на задачах гораздо большей размерности. Что касается конкретных единиц измерения временной сложности алгоритма, то чаще говорят о количестве необходимых элементарных операций типа сложения, умножения, сравнения.

Библиографический комментарий

К настоящему моменту на русском языке имеется достаточно много книг по теории графов, из которых наиболее подходящими для знакомства с общими вопросами этой области математики являются книги Ф. Харари [80], О. Оре [63] и Р. Уилсона [75]. Книга А. А. Зыкова [33] более фундаментальна, но и более трудна для прикладников. Прикладным и алгоритмическим аспектам теории графов, но безотносительно к реализации на ЭВМ посвящены книги [1, 10, 27, 53, 78]. Реализации теоретико-графовых алгоритмов и оценкам трудоемкости уделено достаточно много внимания в книгах [7, 61, 72].

Марковская модель для последовательных программ была предложена Р. Карпом в 1960 г.; см. [40]. Различные граф-модели подробно рассмотрены в обзоре Б. А. Головкина [19], широкий спектр графовых моделей представлен в работе В. Бовье [103], а также в книге А. П. Ершова [30]. Описание граф-модели информационно-логической системы взято из книги [20], пример построения управляющего графа для программы BUBBLE взят из статьи [143].

Различные представления графов описаны в препринте В. К. Попкова [66], различные представления деревьев — в препринте [67], однако большая их часть имеет лишь теоретический интерес. Заслуживает упоминания представление графов в виде K -формул. Это малораспространенное представление описано в книге [10]. Доказательство оптимальности кода Прюфера взято из работы [4].

ГЛАВА 2

ГЛОБАЛЬНЫЙ АНАЛИЗ ГРАФОВ

Под глобальным анализом графа мы понимаем выявление структуры графа и определение характеристик, необходимых для решения задач. Глобальный анализ состоит в сборе и организации информации о строении графа и в выявлении требуемых подструктур с уточнением инфраструктуры графа относительно выделенных подструктур. Наиболее распространенным методом сбора информации является специальным образом организованный обход вершин и дуг (ребер) графа. Информация, получаемая таким способом, оформляется в виде подходящей нумерации вершин.

§ 1. Нумерации, выявляющие логическую структуру графа

1.1. *Нумерацией* F вершин графа G называется инъективное отображение множества вершин $V(G)$ графа в множество N натуральных чисел, $F: V(G) \rightarrow N$. Иногда нумерация понимается в более широком смысле. Это либо отображение вида $F_h: V(G) \rightarrow \underbrace{N \times N \times \dots \times N}_h$ (векторная нумерация), либо

отображение множества вершин в некоторое множество объектов нечисловой природы, например в множество символов некоторого алфавита, множество слов и т. д.

Пусть F — некоторая нумерация. Номер вершины p в нумерации F обозначается $F(p)$, а множество последовательно занумерованных в нумерации F вершин или последовательно F -занумерованных вершин $\{q | q \in V, F(q) \in \{i, \dots, j\}\}$ — через $F[i, j]$ или через $F[u: v]$, где $F(u) = i$ и $F(v) = j$. Дугу (u, v) назовем *обратной дугой* относительно нумерации F , если $F(u) > F(v)$. О верши-

нах p , для которых $F(u) \leq F(p) < F(v)$, будем говорить, что они находятся под обратной дугой.

Путь $\mu = (q = x_{i_0}, x_{i_1}, \dots, x_{i_n} = p)$ называется F -путем, если для каждого $i \in [1:n]$ выполняется условие $F(x_{i-1}) < F(x_i)$. Вершина p называется F -достижимой из вершины q , если существует F -путь из q в p .

Из обширного класса нумераций наиболее важными для логического анализа графа являются нумерации, основанные на поиске в глубину (*базисные нумерации*), прямая нумерация и аранжировка. Эту группу нумераций иногда объединяют под названием *линейные нумерации*.

1.2. Поиск в глубину есть регулярный обход графа, называемый иногда стандартным обходом, по правилам:

1) Находясь в вершине x , нужно двигаться в любую другую, ранее не пройденную вершину (если таковая найдется), одновременно запоминая дугу, по которой мы впервые попали в нее.

2) Если из вершины x мы не можем попасть в ранее не пройденную вершину или таковой вообще нет, то мы возвращаемся в вершину z , из которой впервые попали в x , и продолжаем поиск в глубину из вершины z .

При выполнении обхода графа по этим правилам мы стремимся проникнуть в глубь графа так далеко, как это возможно, затем отступаем на шаг назад и снова стремимся пройти вперед и так далее. При поиске в глубину в орграфе мы можем попасть в вершину y из вершины x только в том случае, если в орграфе есть дуга (x, y) , т. е. мы должны двигаться вперед в направлении ориентации дуг, а возвращаться против ориентации. В неориентированном графе таких ограничений нет.

Отметим несколько свойств поиска в глубину, введя предварительно два простых вспомогательных определения. Будем называть дуги, пройденные при поиске в глубину дважды (в прямом и обратном направлениях), *отмеченными*, а путь $\mu[a, x]$ из начала обхода a в текущую вершину x , не содержащий отмеченных дуг, — *текущим* путем.

Свойство 1. После завершения обхода орграфа при поиске в глубину частичный граф, образованный отмеченными дугами, есть корневое ордеревое с корнем в начале обхода.

Свойство 2. Для того чтобы после завершения обхода орграфа при поиске в глубину все вершины оказались пройденными, необходимо и достаточно, чтобы обход начинался во входе и этот вход был единственным. При этом корневое ордеревое, образованное отмеченными дугами, имеет множество вершин, совпадающее со всем множеством вершин графа, и тем самым является каркасом графа, называемым *деревом поиска в глубину*.

Свойство 3. Текущий путь есть простой путь в орграфе. Его можно продолжить либо до тупикового, т. е. заканчивающегося висячей вершиной, либо до циклического, т. е. до встречи с уже пройденной вершиной.

М-нумерацией вершин графа называется нумерация вершин, соответствующая порядку их обхода при поиске в глубину.

Лемма 1. Для любого простого пути $\mu = [x_1, \dots, x_s]$, такого, что $M(x_1) = \min \{M(x_j) \mid 1 \leq j \leq s\}$, в орграфе найдется *М-путь* из x_1 в x_j для всех $j \in [1:s]$.

Доказательство проведем индукцией по s . Для $s = 1$ утверждение тривиально. Пусть лемма верна для всех путей, состоящих менее чем из s вершин. Рассмотрим путь, состоящий из s вершин. Возможны два случая: $M(x_s) > M(x_{s-1})$ или $M(x_s) < M(x_{s-1})$. В первом случае справедливость леммы вытекает из определения *F-пути* и индуктивного предположения. Во втором случае имеем $M(x_s) < M(x_{s-1})$ и, следовательно, существует такое j , $1 \leq j \leq s-1$, что $M(x_j) < M(x_s) < M(x_{j+1})$. Из определения *М-нумерации* следует, что для любых трех вершин q_1, q_2, q_3 , таких, что $(q_1, q_2) \in E(G)$ и $M(q_1) < M(q_3) < M(q_2)$, существует предшественник q вершины q_3 , для которого $M(q_1) \leq M(q) < M(q_3)$. Например, это может быть вершина, из которой мы впервые попали в q_3 (рис. 2.1). Применяя это замечание вначале к вершинам x_j, x_{j+1}, x_s , убеждаемся в существовании вершины y , из которой x_s *М-достижима* и для которой выполнено соотношение $M(x_j) \leq M(y) < M(x_s)$. Повторное применение этого же замечания, но уже к вершинам x_j, x_{j+1}, y и т. д. в силу конечности графа приводит к нахождению *М-пути* из x_j в x_s .

Следствие. Всякая вершина орграфа с одним входом *М-достижима* из него.

После завершения поиска в глубину все дуги орграфа разбиваются на четыре класса.

1. Класс T *древесных* дуг, порождающих дерево поиска в глубину; для каждой дуги $(x, y) \in T$ имеем $M(x) < M(y)$.

2. Класс F *прямых* дуг, к которому относятся дуги (x, y) такие, что $M(x) < M(y)$ и вершина x соединена с y путем, состоящим из древесных дуг.

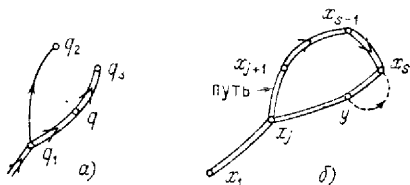


Рис. 2.1.

3. Класс B *обратных* дуг, представляющих собой дуги вида (x, y) такие, что $M(x) > M(y)$ и вершина y соединена с x путем, состоящим из древесных дуг.

4. Класс C *поперечных* дуг, представляющих собой дуги вида (x, y) такие, что $M(x) > M(y)$ и вершины x и y не соединены путем, состоящим из древесных дуг.

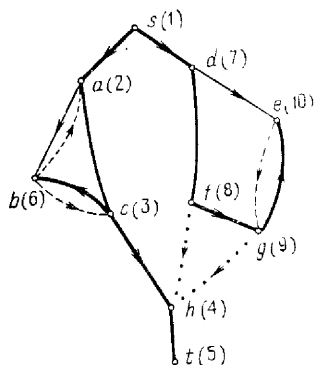


Рис. 2.2.

Основное свойство поперечных дуг — неравенство $M(x) > M(y)$ — доказывается просто. В самом деле, если (x, y) — поперечная дуга, то это означает, что вершина y — конец дуги — была пройдена при поиске в глубину раньше и существует вершина z , в которую мы вернулись, прежде чем пошли в вершину x .

Пример M -нумерации, построенной поиском в глубину, показан на рис. 2.2. Жирными линиями выделены древесные дуги, тонкими линиями — прямые, штриховыми — обратные и пунктиром — поперечные.

1.3. Алгоритм поиска в глубину и построения M -нумерации в ориентированном графе.

Вход: Граф $G = (V, E)$, заданный списками смежности $A(v)$ для $v \in V$, с множеством входных вершин V_0 .

Выход: M -нумерация вершин и разбиение множества E на четыре класса: древесных дуг T , прямых дуг F , обратных дуг B и поперечных дуг C .

начало

процедура ПОИСК (v, N);

начало

1. присвоить вершине v номер $M(v) = N$;

2. $N := N + 1$;

3. для $w \in A(v)$ цикл

4. если вершина w не имеет M -номера то

5. **начало**

6. добавить (v, w) к T ;

7. ПОИСК (w, N);

8. **конец**

9. иначе **начало**

10. если M -номер вершины w больше M -номера вершины v то

11. добавить (v, w) к F ;

12. иначе если существует M -путь из w в v то

13. добавить (v, w) к B ;

14. иначе добавить (v, w) к C ;

15. **конец**

16. **конец**

17. $T \leftarrow \emptyset$; $F \leftarrow \emptyset$; $B \leftarrow \emptyset$; $C \leftarrow \emptyset$; $N := 1$;

18. для $v \in V_0$ цикл

19. пометить вершину v как не имеющую номера;

20. пока существует вершина $v \in V_0$ без M -номера

цикл

21. ПОИСК (v, N);

22. **конец**

1.4. Стеком называется список, у которого элементы добавляются или удаляются только на одном конце списка — *вершине* стека. В другую сторону список имеет неограниченную длину. Другими словами, в стеке элементы вставляются или удаляются по принципу «последний вошел — первый вышел». Часто стек реализуется в виде одного массива, как показано на рис. 2.3. Новый элемент помещается в вершину стека, а все хранящиеся там элементы сдвигаются в глубь стека. Удалить можно только элемент, находящийся в вершине стека, при этом остальные элементы поднимаются на один шаг вверх. Иногда стек обозначает стековую память, когда разрешается считывать содержимое стека без изменения элементов ниже вершины.

Использование стека упрощает реализацию алгоритма поиска в глубину. Присвоение M -номера происходит в тот момент, когда вершина включается в стек; удаление вершины происходит в момент, когда оказываются пройденными все дуги, исходящие из данной вершины.

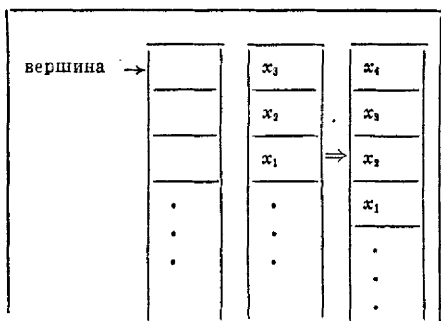


Рис. 2.3.

N -нумерацией вершин называется нумерация, при которой номер n , где n — число вершин в графе, присваивается первой выброшенной из стека вершине, номер $(n-1)$ — второй вершине и т. д. M -нумерация, а следовательно и соответствующая N -нумерация, определяется для данного графа с одной входной вершиной произвольно, но N -нумерация для построенной M -нумерации определяется однозначно. Это иллюстрирует рис. 2.4.

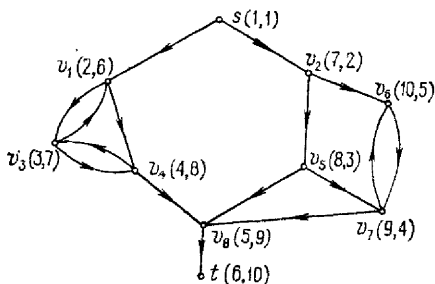


Рис. 2.4.

Из леммы 1 следует, что если s — единственный вход графа, то $N(s) = 1$ и для любой вершины v существует N -путь из s .

1.5. Поиск в глубину в неориентированном графе отличается от поиска в глубину в орграфе тем, что все ребра разбиваются только на три класса: класс древесных ребер, класс ребер касания и класс прямых ребер. Класс ребер касания соответствует классам обратных и поперечных дуг в орграфе. Изменения, которые нужно внести в алгоритм поиска в глубину, очевидны. Поиск в глубину в неориентированном графе превращает исходный граф G в оргграф \vec{G} , индуцируя на каждом ребре ориентацию, совпадающую с направлением прямого прохождения.

Трудоёмкость поиска в глубину определяется, во-первых, тем, что каждая вершина должна быть рассмотрена и ей должен быть приписан номер (это требует $O(n)$ операций для n -вершинного графа), и, во-вторых, тем, что должна быть рассмотрена каждая дуга или каждое ребро (это требует $O(m)$ операций для графа с m дугами). Так как обычно $m > n$, то отсюда следует, что общая трудоёмкость поиска в глубину и в оргграфах, и в графах равна $O(m)$.

1.6. Нумерация F называется *аранжировкой* или *A-нумерацией*, если каждый простой путь из входа s одноходового графа является F -путем. Граф называется *аранжируемым*, если он допускает аранжировку. Пример аранжируемого графа приведен на рис. 2.5, а, пример неаранжируемого — на рис. 2.5, б. На факт неаранжированности указывает наличие в графе двух путей из входа s в вершину v_2 , один из которых содержит вершину v_3 , а второй — нет. Любое отношение порядка между v_2 и v_3 нарушает условие существования аранжировки. Построение аранжировки для ациклического графа проводится по следующему алгоритму.

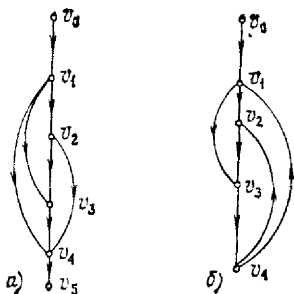


Рис. 2.5.

Вход: Граф $G = (V, E)$, заданный списками смежности $A(v)$ для $v \in V$, с множеством входных вершин V_0 .

Выход: Аранжировка (A-нумерация) вершин графа.

1. начало
2. процедура АРАНЖИРОВКА (v, N);
3. начало
4. присвоить вершине v номер $A(v) = N$;
5. $N := N + 1$;
6. для $w \in A(v)$ цикл
7. если все предшественники вершины w имеют A -номера то
8. АРАНЖИРОВКА (w, N);
9. конец
10. $N := 1$;
11. для $v \in V_0$ цикл
12. пометить вершину v как не имеющую A -номера;
13. пока существует вершина $v \in V_0$ без A -номера
- цикл
14. АРАНЖИРОВКА (v, N);
15. конец

Из N -достижимости любой вершины в графе с одним входом вытекает, что для ациклического (бесконтурного) орграфа аранжировка совпадает с N -нумерацией.

Лемма 2. Пусть $G(p)$ — сильно связный граф, каждый контур которого проходит через выделенную вершину p ; тогда граф $G(p)$ аранжируем с $A(p) = 1$ и аранжировка совпадает с N -нумерацией при поиске в глубину с началом в вершине p .

Для доказательства достаточно заметить, что после удаления дуг, заходящих в вершину p , граф $G(p)$ превращается в бесконтурный.

Следствие. Если граф содержит сильно связный подграф, в который заходят по крайней мере две дуги, то такой орграф неаранжируем.

Теорема 1. Граф аранжируем, если в каждый сильно связный подграф заходит точно одна дуга.

Доказательство. Построим для рассматриваемого графа граф конденсации, или граф Герца, стянув все его бикомпоненты в вершины. Полученный граф является ациклическим и потому аранжируемым. Из леммы 2 следует, что любая система вложенных зон, в каждую из которых заходит только одна дуга, представляет собой аранжируемый граф, аранжировка которого совпадает с N -нумерацией. Комбинируя исходную A -нумерацию графа конденсации и A -нумерации бикомпонент, построим A -нумерацию исходного графа.

1.7. Прямая нумерация. Формальный цикл относительно нумерации F есть подграф, порожденный:

а) вершинами $F[i:k]$ в том и только том случае, если существует обратная дуга (v_k, v_i) и не существует дуги (v_m, v_j) такой, что $i < j \leq k < m$ (такой формальный цикл носит название *примитивного*);

б) вершинами $F[h:k]$, если он либо примитивный формальный цикл, либо если найдутся формальный цикл $F[i:k]$ и некоторая дуга (v_j, v_h) такая, что $h < i \leq j < k$.

Более интуитивное, но менее точное определение формального цикла состоит в следующем. Для каждой обратной дуги (v_k, v_i) существует формальный цикл R с головной вершиной v_i , который строится в соответствии с правилом:

Вначале R есть $F[i:k]$. Если найдется обратная дуга из вершины цикла R в вершину вне R , отличную от v_i , т. е. (v_j, v_h) , $h < i \leq j < k$, то R расширяется за счет включения в него вершины v_h и всех вершин между ними, т. е. полагаем $R = F[h:k]$. Эта процедура повторяется так долго, как это возможно; результирующее множество вершин порождает формальный цикл.

Примеры формальных циклов приведены на рис. 2.6. Заметим, что множество под обратной дугой не обязательно формальный цикл или зона (рис. 2.6, б).

Нумерация F называется *прямой*, если она удовлетворяет условиям: а) каждый формальный цикл есть зона; б) для любой вершины графа существует F -путь из начальной вершины графа в нее; в) для любой обратной дуги (v_k, v_i) существует F -путь из ее начала

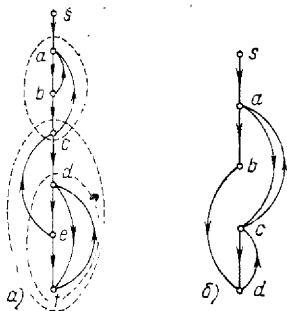


Рис. 2.6.

в каждую вершину v_j , $j \in [1:k]$ (вершину под обратной дугой). Эти условия независимы (рис. 2.7).

Построение прямой нумерации производится на основе M -нумерации.

Вход: Граф $G = (V, E)$, заданный списками смежности $A(v)$ и обратными списками смежности $B(v)$, с построенной M -нумерацией.

Выход: Прямая нумерация (P -нумерация) вершин графа.

начало

1. процедура ПЕРЕНУМЕРАЦИИ (v_i, L_i);
2. начало
3. пока $L_i \neq \emptyset$ цикл
4. выбрать вершину v_i с наименьшим M -номером из L_i ;

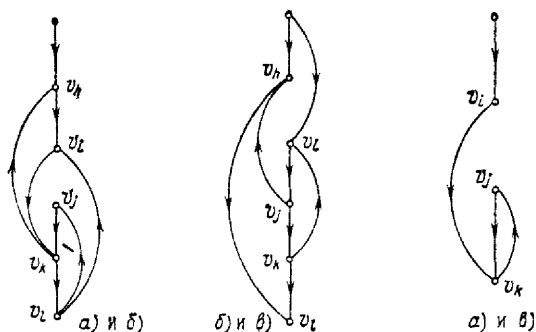


Рис. 2.7.

5. если v_i не помечена то
6. пометить v_i ;
7. иначе переход к шагу 23;
8. для всех помеченных вершин w цикл
9. если $t < k < n \wedge v_k \in B(w)$ то
10. пометить v_k ;
11. продолжать цикл, пока это возможно;
12. $t :=$ наибольший M -номер помеченной вершины;
13. для $u = i + 1$ пока $u < t$ цикл
14. если v_u не помечена то
15. включить v_u в U ;
16. $j := i + 1$;
17. для каждой помеченной вершины w в порядке возрастания их M -номеров цикл
18. $M(w) := j$;
19. $j := j + 1$;
20. для каждой вершины v из U в порядке возрастания их M -номеров цикл
21. $M(w) := j$;
22. $j := j + 1$;
23. $L_i := L_i - \{v_i\}$;
24. конец
25. $i := n$; примечание ($n = |V|$);
26. для $v_i \in B(v_i)$ цикл

27. если $i < j < n$ то
28. $v_i \in L_i$;
29. пока $i > 0$ цикл
30. стереть все метки;
31. ПЕРЕНУМЕРАЦИЯ (v_i, L_i);
32. $i := i - 1$;
33. для $v \in V$ цикл
34. $P(v) := M(v)$;
35. конец

Для данной M -нумерации прямая нумерация определяется однозначно, но обратное неверно. Для доказательства того, что приведенный алгоритм действительно порождает прямую нумерацию, рассмотрим ряд вспомогательных утверждений.

Лемма 3. Если для исходной M -нумерации в графе существовал M -путь из a в b , то тот же самый путь будет и P -путем для прямой нумерации, индуцированной исходной M -нумерацией.

Доказательство. Если в исходном графе выполнялось соотношение $M(a) < M(b)$, то это соотношение выполняется и для индуцированных P -номеров, если только вершина a на шаге 14—15 не попадет в множество U , в то время как вершина b будет помеченной. Но в таком случае не может существовать M -путь из a в b , так как иначе a должна быть помеченной по определению процесса расстановки меток и находиться среди помеченных вершин, а не в U .

Следствие. Для каждой дуги (a, b) разность $P(b) - P(a)$ имеет тот же знак, что и знак соответствующей разности для исходной M -нумерации.

Лемма 4. Алгоритм построения прямой нумерации проверяет каждую вершину в точности один раз на возможность использования ее в качестве начальной вершины контура в порядке, обратном порядку, устанавливаемому M -нумерацией.

Доказательство. В каждый момент работы алгоритма все множество вершин делится на два непересекающихся подмножества $A = \{v \mid M(v) < i\}$ и $B = \{v \mid M(v) \geq i\}$. Изменение нумерации вершин на шагах 17—22 при рассмотрении вершины не изменяет ни упорядочения вершин в A , ни состава множества A ; уменьшение индекса i на шаге 32 просто удаляет вершину с наибольшим M -номером из A и переводит ее в множество B для проверки на возможное начало

контура. Значение i никогда не увеличивается, так что ни одна вершина из B не удаляется.

Для конечного графа алгоритм построения прямой нумерации завершается за конечное число шагов. Это вытекает из того, что при проверке каждой вершины на возможность использования ее в качестве начальной вершины контура проверяется конечное число вершин для каждой обратной дуги, число которых также конечно, и новых обратных дуг при работе алгоритма не появляется.

Использованный термин — начальная вершина контура — отражает семантику алгоритма построения прямой нумерации в том смысле, что изменение нумерации на шагах 17—22 может пониматься как отыскание возможного контура, проходящего через рассматриваемую вершину — начальную вершину потенциального контура, — в виде множества вершин $M[i:t]$, а затем удаление вершин, не входящих фактически в контур, путем простой перенумерации (чистка потенциального цикла).

Лемма 5. Алгоритм построения прямой нумерации включает новую вершину v_k в множество $[v_a:v_c]$ под обратной дугой (v_c, v_a) , $M(v_a) \leq M(v_c)$, только во время чистки некоторого потенциального цикла с начальной вершиной v_b , причем до чистки выполняется соотношение $M(v_a) < M(v_b) < M(v_c) < M(v_k)$.

Доказательство. Если вершина v_k не содержалась до чистки цикла в $[v_a:v_c]$, то либо $M(v_k) < M(v_a)$, либо $M(v_c) < M(v_k)$. Рассмотрим случай, когда (до чистки) $M(v_k) < M(v_a)$. Алгоритм может изменить порядок v_k и v_a только в том случае, если $v_k \in U$ и $v_a \in \bar{M}$ ($v_a \in \bar{M}$ означает, что v_a помечена) для некоторой чистки. Заметим, что если $v_a \in \bar{M}$ во время чистки, то и v_c будет помечена, так как существует дуга (v_c, v_a) , и $M(v_a) \leq M(v_c)$. Таким образом, даже если $v_k \in U$ во время чистки, то результирующий порядок будет тот же: $M(v_a) \leq M(v_c) < M(v_k)$ и v_k не будет включена в множество $[v_a:v_c]$.

Рассмотрим второй случай: $M(v_c) < M(v_k)$, $v_c \in U$ и v_k помечена. Этот случай может дать в результате порядок $M(v_a) < M(v_k) < M(v_c)$. Рис. 2.8 иллюстрирует эту ситуацию. Но если это — окончательный порядок, то $v_a \notin U$ во время чистки (иначе чистка поменяла бы порядок вершин v_a и v_b); v_a также не помечена, а вершины v_a и v_b не совпадают, тогда v_c должна быть по-

мечена. Поэтому $v_a \notin [v_b : v_c]$. Так как $v_c \in U \subset [v_b : v_a]$ и $M(v_a) \leq M(v_c)$, то имеет место окончательно соотношение $M(v_a) < M(v_b) < M(v_c)$.

Теорема 2. В графе с прямой нумерацией, построенной описанным выше алгоритмом, для каждой обратной дуги (v_c, v_a) с $P(v_a) \leq P(v_c)$ существует P -путь из начальной вершины цикла v_a в множество вершин $\{v_k\}$ таких, что $P(v_a) < P(v_k) \leq P(v_c)$.

Доказательство. Утверждение теоремы справедливо для M -нумерации. Лемма 3 показывает, что эти пути одновременно являются и P -путями. Поэтому достаточно показать, что алгоритм не добавляет новых вершин в $[v_a : v_c]$, к которым не существует M -путей из v_a . Доказательство проводится следующим образом: пусть

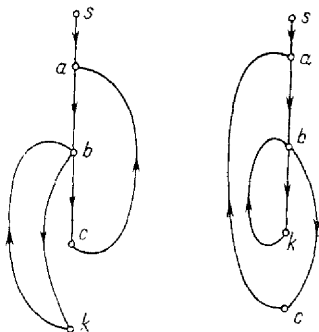


Рис. 2.8.

в начале любой чистки теорема верна; покажем, что чистка не влияет на справедливость теоремы. По лемме 5 любое включение вершины v_k в множество $[v_a : v_c]$ может произойти только тогда, когда происходит чистка цикла с начальной вершиной v_b , такой, что $M(v_a) < M(v_b) < M(v_c) < M(v_k)$; поэтому по предположению у нас есть M -путь из v_a в v_b (так как существует дуга (v_c, v_a) и $M(v_a) < M(v_b) < M(v_c)$). Для чистки потенциального цикла с начальной вершиной v_b и конечной v_i (ее индекс определяется на шаге t) процедура расстановки меток гарантирует существование пути внутри множества $M^* = \bar{M} \cup \{v_b\}$ (\bar{M} — множество помеченных вершин) из каждой помеченной вершины v_m в v_b . Каждый такой путь должен включать обратную дугу (v_y, v_x) , такую, что $M(v_b) \leq M(v_x) \leq M(v_m) \leq M(v_y)$. По предположению либо $v_m = v_x$, либо существует M -путь из v_x в v_m . Но заметим, что v_x сама в M^* , так что либо $v_x = v_b$, либо существует M -путь из некоторой начальной вершины цикла v_z , $M(v_b) \leq M(v_z) < M(v_x)$, в v_x , по тем же соображениям. Ясно, что эта последовательность начальных вершин циклов с уменьшающимися номерами, превышающими (или равными) M -но-

мер вершины v_b , ведет в v_b , откуда следует существование P -пути из v_b к каждой помеченной вершине, включая v_k . Соединяя эти два пути вместе, получаем P -путь из v_a в v_b и в v_k .

Следствие 1. В процессе работы алгоритма ни одна вершина не включается в ранее вычищенный потенциальный цикл, т. е. в множество $[v_a : v_b]$, где v_a — начальная, v_b — конечная вершины для некоторой предыдущей чистки.

Следствие 2. Для ранее вычищенного цикла $[v_a : v_b]$ существует P -путь из v_a к каждой вершине v_k , для которой выполнены неравенства $M(v_a) < M(v_k) \leq M(v_b)$.

Лемма 6. В процессе работы алгоритма не удаляются вершины из ранее вычищенных потенциальных циклов. Другими словами, для данного вычищенного цикла $[v_a : v_c]$ и любой вершины v_b , такой, что $M(v_a) < M(v_b) < M(v_c)$, алгоритм не меняет взаимного порядка вершин v_a , v_b или v_b , v_c .

Теорема 3. Порождаемая алгоритмом нумерация есть прямая нумерация.

Доказательство. Из леммы 3, следствия леммы 1 и теоремы 2 следует, что нумерация, порождаемая алгоритмом, обладает свойствами б) и в) определения прямой нумерации. Остается показать, что каждый формальный цикл $[v_k : v_l]$ есть сильно связный подграф. Как было отмечено ранее, процедура расстановки меток гарантирует существование внутри множества $M^* = \bar{M} \cup \{v_k\}$ M -пути из любой вершины $v_k \in \bar{M}$ к начальной вершине v_k вычищенного потенциального цикла. По следствию 2 теоремы 2 существует M -путь из v_k в любую вершину v_i из \bar{M} . Поэтому M^* есть сильно связный подграф, и, таким образом, $M^* = [v_k : v_l]$ есть вновь вычищенный цикл.

Было уже показано (следствие 1 теоремы 2 и лемма 6), что в процессе работы алгоритма вершины не добавляются и не удаляются из уже вычищенного цикла. Поэтому по окончании работы алгоритма все формальные циклы суть сильно связные подграфы. Остается показать, что каждый формальный цикл был либо вычищенный потенциальный цикл, либо сильно связный подграф. Пусть $[v_a : v_b]$ есть формальный цикл после завершения работы алгоритма. По определению формального цикла существует обратная дуга (v_c, v_a) с $M(v_a) \leq M(v_c) \leq M(v_b)$, которая в силу следствия

леммы 3 и леммы 4 иницирует чистку, если только не выполнены условия: 1) $v_c = v_a$ или 2) v_c помечается, когда проверяется дуга (v_c, v_a) . Случай 1) соответствует наличию петли, которая не оказывает влияния на построение M -нумерации, ни на построение P -нумерации. В случае 2) v_c может быть помечена только тогда, когда она была в $[v_a : v_d]$, т. е. в ранее вычищенном цикле с той же самой начальной вершиной. Чистка цикла $[v_a : v_d]$ могла быть иницирована только при рассмотрении обратной дуги (v_k, v_a) , для которой $M(v_a) < M(v_k) < M(v_c)$ в силу порядка, в котором обрабатываются вершины. Из теоремы 2 и следствия 2 из нее, очевидно, вытекает наличие M -пути из v_k в v_c . Таким образом, если даже все метки будут стерты и процедура расстановки меток начнется снова, но уже с вершины v_c , то v_k будет снова помечена, а также будут помечены вершины из $[v_a : v_d] \setminus \{v_a\}$; другими словами, тот факт, что обратная дуга (v_c, v_a) при ее рассмотрении не иницирует чистку, не может привести к результату, отличному от того, который был бы получен, если бы такая инициализация имела место. Таким образом, мы можем предположить, что по крайней мере $[v_a : v_c]$ было вычищено.

По определению формального цикла либо $[v_a : v_c] = [v_a : v_b]$, либо найдется некоторая дуга (v_g, v_f) , для которой $M(v_a) < M(v_f) \leq M(v_c) < M(v_g)$. Если такая дуга существует после завершения работы алгоритма, она должна была существовать и тогда, когда чистился $[v_a : v_c]$, так как по лемме 6 v_g не могла бы быть удалена после этой чистки, а по следствию 1 теоремы 2 v_f не может быть включена; по следствию леммы 3 обратная дуга порождает неравенство $M(v_f) < M(v_g)$ во время работы алгоритма. Но тогда множество вершин, подвергающихся чистке, должно быть по крайней мере $[v_a : v_g]$. Продолжая те же рассуждения (конечное число шагов), покажем, что подвергнувшийся чистке потенциальный цикл был по крайней мере $[v_a : v_b]$. Так как нет дуги (v_k, v_j) с $M(v_a) < M(v_j) \leq M(v_b) < M(v_k)$, то по определению формального цикла не возникает ситуации, при которой бы чистился цикл $[v_a : v_b]$. Таким образом, вычищенный цикл был в точности $[v_a : v_b]$. Теорема доказана.

1.8. F -областью $F[p]$ вершины p для некоторой нумерации F , такой, что $F(p) = i$, называется множество

тех вершин графа G , из которых достижима вершина p в подграфе, порожденном вершинами из $F[i:n]$.

Пусть фиксирована некоторая M -нумерация и связанная с ней N -нумерация. Вершина v называется *бивершиной*, если для любого индекса $i < N(v)$ имеем $v \in N[i]$.

Пример. Для графа с N -нумерацией на рис. 2.4 имеем $N[v_1] = \{v_1\}$, $N[v_2] = \{v_2\}$, $N[v_3] = \{v_3\}$, $N[v_4] = \{v_4, v_5\}$, $N[v_5] = \{v_5\}$, $N[v_6] = \{v_6, v_7, v_8\}$, $N[v_7] = \{v_7, v_8\}$, $N[v_8] = \{v_8\}$, $N[v_9] = \{v_9\}$, $N[v_{10}] = \{v_{10}\}$. В этом графе бивершинами являются вершины $v_1, v_2, v_3, v_4, v_6, v_8, v_9, v_{10}$.

T -нумерацией называется нумерация, обладающая следующими свойствами:

а) для любых двух бивершин p и q соотношение $T(p) < T(q)$ выполняется тогда и только тогда, когда $N(p) < N(q)$;

б) для любой бивершины p имеем

$$N[p] = T[T(p): T(p) + |N[p]| - 1].$$

Так как для построения T -нумерации необходимо уметь строить множество F -областей, опишем алгоритм построения F -областей в виде процедуры-функции ОБЛАСТЬ (F, p).

функция ОБЛАСТЬ (F, p);

начало

1. включить p в пустое множество A ;
2. включить p в пустое множество B ;
3. пока $A \neq \emptyset$ цикл
4. выбрать и удалить любую вершину q из A ;
5. для всех w из $\Gamma^{-1}(q)$ цикл
6. если $F(w) > F(p) \wedge w \notin B$ то
7. добавить w в B ;
8. добавить w в A ;
9. конец цикла;
10. конец цикла;

конец

Область (F, p) определяется множеством вершин B . Алгоритм построения T -нумерации:

Вход: Граф $G = (V, E)$ с N -нумерацией вершин.

Выход: T -нумерация вершин и множество BV бивершин.

начало

1. $BV := \emptyset$, $T1 := 1$;
2. для k от 1 до n шаг 1 цикл

3. выбрать вершину z , у которой $N(z) = k$;
 4. если z не имеет T -номера то
 5. положить $T(z)$ равным $T1$;
 6. $T1 := T1 + 1$;
 7. добавить z к множеству BV ;
 8. положить A равным ОБЛАСТЬ $(N, z) \setminus \{z\}$;
 9. пока $A \neq \emptyset$ цикл
 10. выбрать и удалить некоторую вершину v из A ;
 11. положить $T(v)$ равным $T1$;
 12. $T1 := T1 + 1$;
 13. конец цикла;
 14. конец цикла;
- конец

Пример графа с N - и T -нумерацией приведен на рис. 2.9.

Поскольку каждая вершина графа принадлежит в точности одной выделенной N -области, то трудоемкость построения T -нумерации равна $O(m)$, где m — число дуг.

1.9. K -нумерация определяется как последний член последовательности $K_1 = N, K_2, \dots, K_n = K$, в которой для любого $i \in [1 : n - 1]$ имеем следующее:

1) $K_{i+1}(p) = K_i(p)$ для каждой вершины $p \in K_i[1 : i - 1]$;

2) для любых двух вершин $p, q \in K_i[i : n]$ справедливо неравенство $K_{i+1}(p) < K_{i+1}(q)$ тогда и только тогда, когда выполнено одно из свойств:

- а) $p \in K_i[i]$ и $q \notin K_i[i]$;
- б) $K_i(p) < K_i(q)$ и либо $\{p, q\} \subseteq K_i[i]$, либо $p \notin K_i[i]$ и $q \notin K_i[i]$.

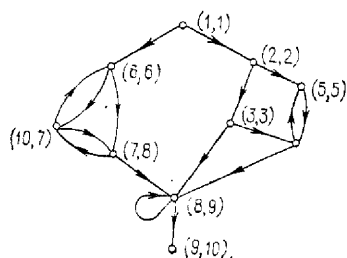


Рис. 2.9.

Алгоритм построения K -нумерации:

Вход: Граф $G = (V, E)$ с K_i -нумерацией вершин.

Выход: K_{i+1} -нумерация.

начало

1. выбрать вершину z такую, что $K_i(z) = i$;
2. положить множество A равным ОБЛАСТЬ (K_i, z) ;
3. $M1 := i$; $M2 := i + |A|$;
4. для k от 1 до n шаг 1 цикл
5. выбрать вершину u такую, что $K_i(u) = k$;

6. если $k < i$ то
7. положить $F1(u)$ равным k ;
8. иначе если $u \in A$ то
9. положить $F1(u)$ равным $M1$;
10. $M1 := M1 + 1$;
11. иначе положить $F1(u)$ равным $M2$;
12. $M2 := M2 + 1$;
13. конец цикла;
- конец

В качестве K_{i+1} -нумерации берется нумерация $F1$.

1.10. Назовем F -линией в некотором множестве вершин $[i:j]$ множество тех вершин подграфа $F[i:j]$, из которых F -достижимы его выходные вершины. F -рангом $R(p)$ вершины p в $[i:j]$ назовем величину $\max(i, \{F(p) \mid p \in A\})$, где A — множество тех вершин F -линии в $[i:j]$, из которых F -достижима вершина q .

L -нумерация есть последний член последовательности $L_n = K, L_{n-1}, \dots, L_0 = I$, в которой для каждого $i \in [1:n]$ имеем следующее:

1) $L_{i-1}(p) = L_i(p)$ для любой вершины p , не входящей в $L_i[i:i + |L_i[i]| - 1]$;

2) для любых двух вершин p и q из $L_i[i:i + |L_i[i]| - 1]$ неравенство $L_{i-1}(p) < L_{i-1}(q)$ будет иметь место тогда и только тогда, когда выполнено одно из свойств:

а) p и q имеют один и тот же L_i -ранг в $[i:i + |L_i[i]| - 1]$, и имеет место неравенство $L_i(p) < L_i(q)$;

б) L_i -ранг вершины q в $[i:i + |L_i[i]| - 1]$ превышает соответствующий L_i -ранг вершины p .

Алгоритм построения L_{i-1} -нумерации по данной L_i -нумерации:

Вход: Граф $G = (V, E)$ с построенной L_i -нумерацией.

Выход: L_{i-1} -нумерация для каждого $t \in [1:n]$.

начало

1. процедура РАНГ(F, i, j);

2. пусть $L1$ есть F -линия для $F[i:j]$;

3. для всякой вершины v из $F[i:j]$ положить $R(v)$ равным t ;

4. для s от j до i через -1 цикл

5. выбрать вершину v , для которой $F(v)$ равен s ;

6. если $v \in L1$ то

7. для всех w из $B(v)$ цикл

8. если $i \leq F(w) < F(v)$ то

9. добавить w в множество L_1 ;
10. **конец** цикла;
11. **конец** цикла;
12. для s от i до j через 1 цикл
13. выбрать вершину v , для которой $F(v)$ равен s ;
14. если $v \in L_1$ то
15. положить $R(v)$ равным $F(v)$;
16. для всех w из $A(v)$ цикл
17. если $F(v) < F(w) \leq j$ то
18. положить $R(w)$ равным максимуму из $R(v)$
и $R(w)$;
19. **конец** цикла;
20. **конец** цикла;
21. **возврат** R ;
22. выбрать вершину u , у которой $L_i(u)$ равен i ;
23. $j := i + |\text{ОБЛАСТЬ}(L_i, u)| - 1$;
24. положить R равным $\text{РАНГ}(L_i, i, j)$;
25. для каждого s из $[i : j + 1]$ положить $\text{ЧИСЛО}(s)$
равным нулю;
26. для всех v из $L_i[i : j]$ цикл
27. положить $\text{ЧИСЛО}(R(v) + 1)$ равным $\text{ЧИСЛО}(R(v)$
 $+ 1) + 1$;
28. **конец** цикла;
29. для s от i до j шаг 1 цикл
30. установить $\text{ЧИСЛО}(s + 1)$ равным $\text{ЧИСЛО}(s + 1)$
 $+ \text{ЧИСЛО}(s)$;
31. **конец** цикла;
32. для s от 1 до n шаг 1 цикл
33. выбрать вершину v , у которой $L_i(v)$ равен s ;
34. если $s \in [i : j]$ то
- примечание** в данный момент для любого $t \in [i : j]$ $\text{ЧИСЛО}(t)$ обозначает количество тех вершин v из $L_i[i : j]$, у которых либо $R(v) < t$, либо $R(v) = t$ и $L_i(v) < s$ **конец** примечания
35. установить $L_i^1(v)$ равным $i + \text{ЧИСЛО}(R(v))$;
36. положить $\text{ЧИСЛО}(R(v))$ равным $\text{ЧИСЛО}(R(v))$
 $+ 1$;
37. **иначе** установить $L_i^1(v)$ равным $L_i(v)$;
38. **конец** цикла;
39. в качестве L_{i-1} взять L_i^1 ;
- конец**

Трудоёмкость алгоритма складывается из трудоёмкости вычисления i и R , что составляет $O(m)$, и трудоёмкости переписывания L_i -номеров, что составляет

$O(n)$. В итоге трудоемкость алгоритма есть $O(m)$, а построение L -нумерации в целом имеет трудоемкость $O(nm)$.

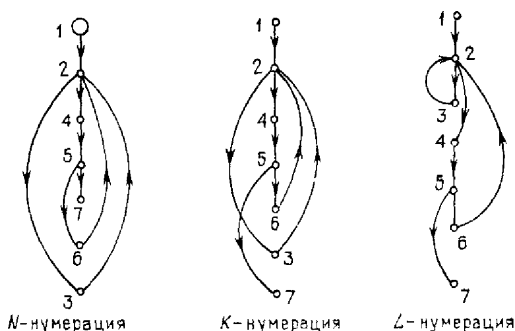


Рис. 2.10.

На рис. 2.10 изображен граф, у которого $L_7 = L_6 = L_5 = L_4 = L_3 = L_2 = K$, $L_1 = L_0 = L$, $L_2[L_2(v_2)] = \{v_2, v_3, v_4, v_5, v_6\}$; следовательно, $L_2[2:6] = \{v_2, v_3, v_4, v_5, v_6\}$.

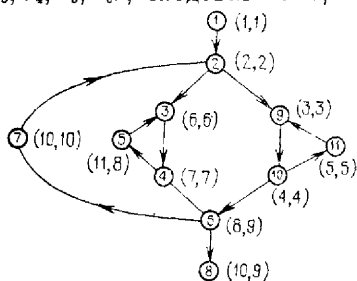


Рис. 2.11.

Здесь L_2 -линия в $[2:6]$ образована вершинами v_2, v_4 и v_5 . Вершины v_2 и v_3 имеют один и тот же L_2 -ранг в $[2:6]$, так же как и вершины v_5 и v_6 .

Другой пример графа и возможной L -нумерации его вершин приведен на рис. 2.11. Вершины графа помечены M -номерами, N -номера и L -номера указаны в скобках (первым N -номер, вторым L -номер) рядом с соответствующими вершинами.

§ 2. Логический анализ управляющих графов.

Линейные компоненты и компоненты сильной связности

2.1. Матрица достижимости и транзитивное замыкание. В том случае, когда необходимо установить лишь факт наличия путей между вершинами, возникает задача о построении транзитивного замыкания орграфа.

Транзитивным замыканием G^* графа G называется орграф с тем же множеством вершин, что и G , но в котором дуга из x_i в x_j идет тогда и только тогда, когда вершина x_j достижима в G из x_i .

Матрицей смежности транзитивного замыкания G^* орграфа G служит матрица достижимости $R(G)$ графа G . Простейший, но не самый эффективный из известных алгоритмов построения матрицы достижимости состоит в последовательном изменении строк матрицы смежности до получения строки матрицы достижимости.

Вход: Матрица $A(G)$ смежности орграфа G , состоящая из строк A_1, A_2, \dots, A_n .

Выход: Матрица достижимости $R(G)$, состоящая из строк R_1, R_2, \dots, R_n .

начало

1. для i от 1 до n шаг 1 цикл

2. сформировать множество J индексов j таких, что $a_{ij} = 1$;

3. $R_i := A_i$; $K := \emptyset$;

4. пока $J \neq \emptyset$ цикл

5. выбрать $j \in J$;

6. $R_i := R_i \oplus A_j$;

7. $J := J \setminus \{j\}$;

8. $K := K \cup \{j\}$;

9. сформировать множество J , индексов k таких, что $a_{jk} = 1$;

10. $J := J \cup (J_j \setminus K)$;

11. конец цикла;

12. возврат R_i ;

13. конец цикла;

конец

Пример. Для графа G на рис. 2.12 матрица достижимости R имеет вид

$$R(G) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix},$$

а его транзитивное замыкание G^* показано на рис. 2.13.

Более эффективным является алгоритм Уоршалла (Warshall). Этот алгоритм находит матрицу R путем вычисления последовательности квадратных матриц

B^0, B^1, \dots, B^n порядка n в соответствии со следующими правилами:

начало

1. $B^0 := A(G)$;

2. для l от 1 до n шаг 1 цикл

3. $B_{ij}^l = B_{ij}^{l-1} \oplus [B_{ii}^{l-1} \& B_{jj}^{l-1}]$;

4. **конец цикла**;

5. $R := B^n$;

конец

Если придать значениям B_{ij}^l надлежащий смысл, то легко убедиться, что алгоритм работает правильно. Действительно, мы хотим, чтобы B_{ij}^l было равно 1

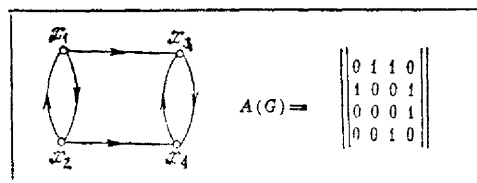


Рис. 2.12.

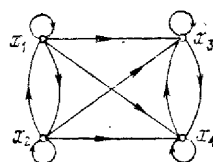


Рис. 2.13.

тогда и только тогда, когда вершины x_i и x_j связаны путем через промежуточные вершины только из множества $\{x_1, x_2, \dots, x_l\}$. Заметим, что начальный шаг $B^0 := A(G)$ (путь без промежуточных вершин) и конечный шаг $R := B^n$ (путь через любые промежуточные вершины) согласуются с таким пониманием смысла B_{ij}^l . Формула же итеративного шага выражает только тот факт, что для x_i и x_j , связанных путем, промежуточные вершины которого принадлежат множеству $\{x_1, \dots, x_l\}$, необходимо и достаточно выполнение одного или обоих условий:

1. Существует путь от x_i до x_j , все промежуточные вершины которого принадлежат множеству $\{x_1, \dots, x_{l-1}\}$.

2. Существуют пути от x_i до x_l и от x_l до x_j , все промежуточные вершины которых принадлежат множеству $\{x_1, \dots, x_{l-1}\}$.

2.2. Матричный алгоритм отыскания бикомпонент орграфа. Знание матрицы достижимости позволяет выявить все компоненты сильной связности в орграфе. Определим *поэлементное*, или *адамарово*, *произведение*

матриц $B = \|b_{ij}\|$ и $C = \|c_{ij}\|$ по правилу: $B \times C = \|b_{ij} \times c_{ij}\|$. Тогда вершины бикомпоненты орграфа, содержащей вершину x_i , определяются единичными элементами i -й строки матрицы $R \times R^T$, где R^T — транспонированная матрица достижимости. Отсюда следует, что, вычислив поэлементное произведение матриц R и R^T и разбив все ненулевые строки этого произведения на группы одинаковых строк, мы можем определить множества вершин каждой бикомпоненты, поскольку номера строк однозначно определяют номера вершин, входящих в бикомпоненту.

2.3. Более эффективные алгоритмы отыскания бикомпонент основаны на обходе графа, реализующего поиск в глубину. В основе их лежат утверждения, в формулировке которых будем предполагать, что бикомпонента может состоять из одной вершины.

Лемма 7. Если $\mu = [v_{i_1}, \dots, v_{i_k}]$ — тупиковый путь в графе G , то:

- а) $\{v_{i_k}\}$ — бикомпонента графа G ;
- б) множество бикомпонент орграфа $G' = (X \setminus \{v_{i_k}\}, U')$, получаемого из G удалением вершины $\{v_{i_k}\}$ и смежных с ней дуг, отличается от множества бикомпонент графа G в точности на бикомпоненту $\{v_{i_k}\}$;
- в) $\mu = [v_{i_1}, \dots, v_{i_k}]$ — простой путь в G .

Лемма 8. Если $\mu = [v_{i_1}, \dots, v_{i_k}]$ — простой путь в G и $v_{i_k} = v_{i_s}$ для $s < k - 1$, то

- а) контур $C = [v_{i_s}, v_{i_{s+1}}, \dots, v_{i_k} = v_{i_s}]$ принадлежит какой-нибудь бикомпоненте;
- б) отождествление всех вершин контура C не меняет множества бикомпонент;
- в) путь $\mu = [v_{i_1}, \dots, \tilde{v}_{i_s}]$, где \tilde{v}_{i_s} — вершина, получившаяся в результате отождествления вершин контура C , есть простой путь.

Вход: Граф $G = (V, E)$, заданный списками смежности $A(v)$.

Выход: Список B бикомпонент.

начало

1. процедура БИКОМП(End (TT));
2. начало
3. $v := \text{End}(TT)$;
4. для $w \in A(v)$ цикл
5. если $A(v) \neq \emptyset$ то

6. если $S(w) = 0$ то
 7. $TT \leftarrow w$;
 8. $S(w) := 1$;
 9. $\text{БИКОМП}(w)$;
 10. иначе примечание вершина w находится в стеке TT конец примечания
 11. стянуть хвост стека TT , начиная с вершины w , в вершину \tilde{w} , т. е. удалить, начиная с вершины w , хвост стека TT , запомнить удаленные вершины и вместо вершины w занести в стек вершину \tilde{w} ;
 12. взять в качестве новых списков смежности объединение списков смежности стянутых вершин;
 13. $\text{БИКОМП}(\tilde{w})$;
 14. иначе удалить v из стека TT и занести v в список B бикомпонент;
 15. $\text{БИКОМП}(\text{End}(TT))$;
 16. конец цикла;
 17. возврат B ;
 18. конец процедуры БИКОМП ;
 19. $TT :=$ пустой стек;
 20. для каждой $v \in V$ цикл положить $S(v)$ равным нулю;
 21. выбрать вершину v_0 с $S(v_0) = 0$;
 22. $TT \leftarrow v_0$;
 23. $S(v_0) := 1$;
 24. $\text{БИКОМП}(v_0)$;
- конец

Трудоёмкость алгоритма равна $O(|E| + |V| \log |V|)$, так как он представляет собой уточнение поиска в глубину, в ходе реализации которого требуются некоторые дополнительные операции при прохождении каждой дуги. Кроме того, каждая вершина один раз включается в стек и в точности один раз исключается из него.

Пример. Для графа G на рис. 2.14 матрицы смежности и достижимости показаны на рис. 2.15. Матрица $M = R \times R^T$ и список B бикомпонент приведены на рис. 2.16. Последовательные фазы отыскания бикомпонент с помощью описанного алгоритма показаны на рис. 2.17.

2.4. Введенные в § 1 различные типы нумерации вершин орграфа, в частности N -нумерация, позволяют выделить подграфы различного вида, в том числе и бикомпоненты.

Лемма 9. Для любого $i \in [1:n]$ подграф $N[i]$ является сильно связным.

Доказательство. Пусть $N[i]$ отличен от тривиального подграфа и p — его вершина, N -номер которой равен i . Рассмотрим вершину q из $N[p]$, отличную от p , и предположим, что $M(q) > M(p)$. Тогда из определения F -области и определения M -нумерации вытекает

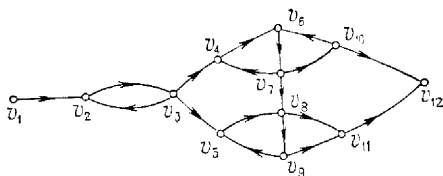


Рис. 2.14.

$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad R(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Рис. 2.15.

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} B_1 &= \{v_1, v_3\}, \\ B_2 &= \{v_5, v_8, v_9\}, \\ B_3 &= \{v_6, v_7, v_{10}\}. \end{aligned}$$

Рис. 2.16.

существование M -пути из p в q . По определению N -нумерации это свидетельствует о существовании N -пути из p в q , который принадлежит $N[i]$. Вместе с тем

из определения F -области следует также существование пути из q в p в подграфе $N[i]$. Таким образом, для доказательства леммы достаточно показать, что $M(p) < M(q)$. Предположим, что это не так, и рассмотрим путь $P = [p_1 = q, p_2, \dots, p_s = p]$ в $N[i:n]$.

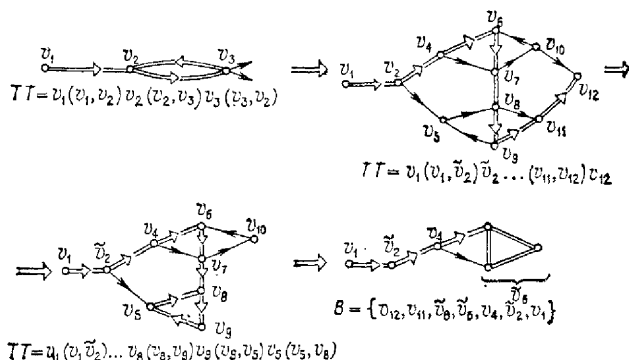


Рис. 2.17.

По нашему предположению $M(p) > M(q) \geq M(p_i) = \min \{M(p_k) \mid k \in [1:s]\}$, и, следовательно, в силу леммы 1 существует M -путь из p_i в p , где $p_i \in N[p]$ и $p_i \neq p$. По определению N -нумерации это означает, что $N(p_i) < N(p)$, что противоречит сделанному предположению.

Следствие 1. Для любой вершины $p \in V$ имеем $M(p) = \min \{N(q) \mid q \in N[p]\}$, и каждая вершина из $N[p]$ является M -достижимой из вершины p .

Следствие 2. Для любых двух различных вершин p и q реализуется одна и только одна из трех возможностей: $M[p] \subset N[q]$, $N[q] \subset N[p]$ или $N[p] \cap N[q] = \emptyset$.

Теорема 4. Подграф S является бикомпонентой орграфа G тогда и только тогда, когда существует бивершина p , для которой $S = N[p]$.

Доказательство. Пусть S — бикомпонента орграфа G , и пусть p — такая вершина из S , что $N(p) = \min \{N(q) \mid q \in S\}$. По определению F -области имеем $S \subseteq N[p]$. Однако по лемме 9 и следствию 2 из нее $N[p] \subseteq S$, откуда следует, что p — бивершина.

Пусть p — бивершина. По лемме 9 некоторая бикомпонента графа (скажем, S) содержит все вершины из $N[p]$. Пусть $S \setminus N[p]$ не пусто. Тогда должна существо-

вать такая вершина q , что $N(q) < N(p)$ и q достижима из вершины p в $N[N(q) : n]$. Это значит, что $p \in N[q]$, и, таким образом, имеет место противоречие.

2.5. Графом Герца (конденсацией, фактор-графом) G^* графа G называется орграф, вершинами которого служат бикомпоненты B_1, \dots, B_r орграфа G и те вершины v , которые не принадлежат ни одной бикомпоненте, а дуга идет из B_i в B_j , если в орграфе G имеется хотя бы одна дуга, ведущая из некоторой вершины бикомпоненты B_i в вершину бикомпоненты B_j , и т. п. Пример орграфа G и его графа Герца приведен на рис. 2.18.

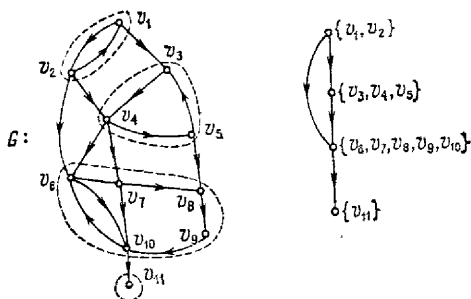


Рис. 2.18.

Из максимальности бикомпонент следует, что в графе Герца нет контуров. В противном случае объединение множеств вершин тех бикомпонент графа G , которые представляют собой вершины контура в его графе Герца, порождало бы в G сильно связный подграф, содержащий вершины из разных бикомпонент, что невозможно.

Будем называть орграф *бесконтурным*, если он не содержит контуров. В литературе для обозначения таких орграфов употребляется также термин «ациклический граф». В последнее время в англоязычной литературе все чаще стал попадаться термин «dag» — аббревиатура словосочетания «directed acyclic graph».

Рассмотрим последнюю вершину некоторого максимального пути бесконтурного орграфа G . Так как для этой вершины нет вершин, достижимых из нее, поскольку либо нашелся бы контур, либо путь не был бы максимальным, то заключаем, что бесконтурный орграф содержит по крайней мере одну вершину с нулевой полустепенью исхода. Рассуждая аналогично,

убеждаемся, что бесконтурный орграф содержит по крайней мере одну вершину с нулевой полустепенью захода.

Теорема 5. *Следующие свойства орграфа G эквивалентны:*

1) G — бесконтурный орграф;

2) G^* изоморфен G ;

3) каждый путь в G простой;

4) вершины орграфа G можно упорядочить таким образом, что матрица смежности $A(G)$ будет верхней треугольной матрицей.

Доказательство теоремы предоставляется читателю.

Указанное в теореме упорядочение вершин носит название *топологической сортировки вершин* и может рассматриваться как процесс отыскания линейного порядка, в который может быть вложен данный частичный порядок. Топологическая сортировка полезна при анализе схем действия, где большой и сложный план представляется как орграф, вершины которого соответствуют целям плана, а дуги — действиям. Топологическая сортировка дает порядок, в котором могут быть достигнуты цели. Топологическая сортировка начинается с отыскания вершины графа G , из которой не выходят дуги, и присвоения этой вершине наибольшего номера, а именно $n = |V|$. Эта вершина удаляется из графа G вместе с входящими в нее дугами. Поскольку оставшийся орграф также бесконтурный, мы можем повторить процесс и присвоить следующий номер $n - 1$ вершине, из которой не выходят дуги, и т. д. Этот процесс полностью совпадает с процедурой построения N -нумерации. Другими словами, топологическая сортировка есть N -нумерация бесконтурных графов.

2.6. Эффективные алгоритмы построения транзитивного замыкания. Построение алгоритмов основано на переходе от орграфа к его графу Герца. В качестве начального шага отыскивается множество бикомпонент и строится граф Герца. Затем строится транзитивное замыкание графа Герца, которое дополняется до транзитивного замыкания исходного графа дугами, инцидентными каждой вершине внутри бикомпоненты.

Для иллюстрации работы такого алгоритма возьмем орграф, показанный на рис. 2.19, а (его граф Герца показан на рис. 2.19, б).

В графе Герца вершины v_1 , v_3 и v_4 соответствуют вершинам a , e и f исходного графа, а вершина v_2 соот-

ветствует бикомпоненте $\{b, c, d\}$. Вначале граф Герца топологически сортируется, т. е. для его вершин определяются N -номера. Транзитивное замыкание G^* отыскивается, начиная с вершины с наибольшим N -номером. Оно формируется в виде перечня составных вершин следующим образом: берется некоторая вершина v_i , еще не включенная в состав составных вершин, и включается в ту из них, которая содержит все концы дуг, исходящих из v_i , т. е. все множество Γv_i ; если таковой нет, то образуется новая составная вершина по этому же принципу. Для графа на рис. 2.19, б замыкание строится последовательно так: (4) , $((3, 4)$, $(2, 4))$, $((1, 2, 4)$, $(1, 3, 4))$, что дает в результате матрицу

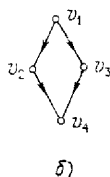
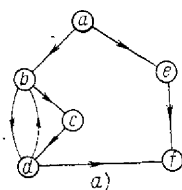


Рис. 2.19.

есть в ту из них, которая содержит все концы дуг, исходящих из v_i , т. е. все множество Γv_i ; если таковой нет, то образуется новая составная вершина по этому же принципу. Для графа на рис. 2.19, б замыкание строится последовательно так: (4) , $((3, 4)$, $(2, 4))$, $((1, 2, 4)$, $(1, 3, 4))$, что дает в результате матрицу

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Транзитивное замыкание исходного графа получается добавлением дуг между всеми вершинами, включенными в составную. В худшем случае алгоритм требует $O(n^3)$ операций.

Более эффективный и наиболее распространенный алгоритм Мунро строит вначале транзитивное замыкание графа Герца, но существенно использует метод Штрассена быстрого перемножения матриц. Его трудоемкость равна $O(n_G^k \log_2 n_G)$, где n_G — число вершин в графе Герца, а k берется из интервала $[2, \log_2 7]$.

Пусть \bar{G} есть граф Герца графа G , и пусть \bar{G} топологически отсортирован, так что его матрица смежности есть верхняя треугольная матрица, у которой на главной диагонали стоят единицы. Если в \bar{G} найдется путь из вершины x_i в x_{i+l} , то он может проходить только через вершины $x_i, x_{i+1}, \dots, x_{i+l}$. Другими словами, для любых k последовательно взятых вершин любой путь между ними лежит внутри порожденного ими подграфа. Основная идея алгоритма Мунро состоит в следующем. Если известны транзитивные замыкания

двух «смежных» подграфов с 2^i вершинами каждый, то может быть построено транзитивное замыкание подграфа с 2^{i+1} вершинами, получающегося объединением исходных 2^i -вершинных подграфов. На рис. 2.20 показана матрица, на которой производятся все описанные вычисления. Если вершина в подграфе 2 достижима из некоторой вершины подграфа 1, то она может быть достигнута по указанной матрице самое большее в три приема. Вначале мы должны попасть в некоторую вершину в подграфе 1, затем в некоторую вершину в подграфе 2 и только потом в нужную вершину из подграфа 2. Иногда первый или третий шаг или оба могут отсутствовать. Построение транзитивного замыкания производится путем последовательного вычисления степеней матрицы, показанной на рис. 2.20, с заменой на каждом шаге ненулевых элементов единицами. При этом все используемые нами матрицы треугольные и все вычисляемые элементы лежат в правом верхнем квадранте.



Рис. 2.20.

Алгоритм Мурро:

Шаг 1. Объединяем вершины графа в пары $(2l - 1, 2l)$ для всех $l \in [1 : m/2]$.

Шаг 2. Строим транзитивные замыкания подграфов.

Шаг 3. Объединяем полученные подграфы попарно, сохраняя монотонность нумерации вершин.

Шаг 4. Выполняя подходящим образом умножение матриц, строим транзитивные замыкания объединенных подграфов.

Повторяем шаги 3 и 4, пока не получим решения задачи для графа Герца.

2.7. Пусть Y, Z — некоторые подмножества вершин орграфа G с входом s .

Замыканием Y относительно Z называется такое множество вершин $M(Y, Z)$ графа G , что $x_i \in M(Y, Z)$, $i \in [1:n]$, если в G найдется путь от некоторой вершины $x_j \in Y$ к вершине x_i , в котором нет вершин из Z . Если Y или Z состоит из одной вершины, то говорят о замыкании вершины или о замыкании относительно вершины.

Обратным замыканием Y относительно Z называется такое множество вершин $N(Y, Z)$ графа G , что $x_i \in N(Y, Z)$, $i \in [1:n]$, если в G найдется путь от вершины x_i к некоторой вершине $x_j \in Y$, в котором нет вершин из Z .

Алгоритм построения замыкания относительно множества вершин:

Вход: Граф $G(V, E)$, заданный матрицей смежности $A(G)$, состоящей из строк $A(1), A(2), \dots, A(n)$; Y, Z, M — строки из нулей и единиц, соответствующие множествам $Y, Z, M(Y, Z)$ (равенство i -го разряда единице означает, что вершина x_i принадлежит данному множеству); булевские операции над строками выполняются поразрядно; строка, соответствующая пустому множеству, обозначается \emptyset .

Выход: Множество $M(Y, Z)$.

начало

1. $M_0 := \emptyset$;

2. $P_0 := Y$;

3. для k от 1 до n шаг 1 цикл

4. начало

5. $E_k := T_k \& \overline{M}_{k-1}$;

6. $M_k := M_{k-1} \oplus E_k$;

7. $P_k := E_k \& \overline{Z} \& \overline{Y}$;

8. конец цикла;

примечание здесь T_k есть дизъюнкция строк $A(j)$ для всех значений j , совпадающих с номерами единичных разрядов строки P_{k-1} , конец примечания

конец

Замечание 1. Фактически алгоритм заканчивается, когда для некоторого $r \leq n$ строка P_r не станет нулевой; тогда $M = M_r$.

З а м е ч а н и е 2. Из описания алгоритма следует, что при построении $M(x_i, \emptyset)$ каждое множество E_k состоит из таких вершин x_i , что кратчайший путь из x_i в x_i содержит k дуг.

З а м е ч а н и е 3. Построение множества $N(Y, Z)$ производится с помощью того же алгоритма, но в качестве A берется A^T .

З а м е ч а н и е 4. Для построения транзитивного замыкания графа G достаточно вычислить $M(s, \emptyset)$, где s — вход G .

Трудоёмкость алгоритма равна $O(ns)$, где s — число ячеек памяти, необходимых для запоминания n двоичных разрядов.

2.8. Нахождение последовательности вложенных зон. Множество зон D называется *иерархией зон* орграфа, если:

1) для любых двух различных зон S_1 и S_2 из D имеет место либо $S_1 \subset S_2$, либо $S_2 \subset S_1$, либо $S_1 \cap S_2 = \emptyset$;

2) для любой зоны S графа G существует такая зона $S_i \in D$, что $S \subseteq S_i$ и зоны S и S_i имеют общую входную вершину.

Опишем матричный алгоритм выявления иерархии зон. Иерархию зон будем изображать матрицей $P = \|p_{ij}\|$ размера $k \times n$, где k — глубина иерархии, т. е. число вложенных зон, а n — число вершин в орграфе. Элемент p_{ij} равен 1, если вершина v_j входит в i -ю зону (для простоты рассматривается случай, когда орграф содержит лишь одну бикомпоненту, изменения для общего случая тривиальны), и равен 0 в противном случае. Матрица P строится построчно.

Вход: Граф G , заданный матрицей смежности $A(G)$.

Выход: Матрица P .

начало

1. для j от 1 до n шаг 1 цикл

2. если $a_{jj} = 1$ то

3. $p_{1j} := 1$;

4. иначе $p_{1j} = 0$;

примечание если $a_{ij} = 1$, то в вершине v_i имеется петля, поэтому вершина v_i составляет множество вершин наименьшей зоны **конец примечания**

5. $D_1 := A$;

6. $l := 2$;

7. $\tilde{A} := A$;

8. пока $P_l \neq P_{l-1}$ цикл

9. $\bar{A} := A \times \bar{A}$;
10. для i от 1 до n шаг 1 цикл
11. для j от 1 до n шаг 1 цикл
12. если $\bar{a}_{ij} < 0$ то
13. $a_{ij}^{(l)} := 1$;
14. если $d_{ij}^{(l-1)} = 0$ то $d_{ij}^{(l)} := l$
15. иначе $d_{ij}^{(l)} := 0$;
16. иначе $a_{ij}^{(l)} := 0$;
17. если $a_{jj}^{(l)} = 1 \wedge d_{ij}^{(l)} > 0 \wedge d_{ji}^{(l)} > 0 \wedge d_{ij}^{(l)} + d_{ji}^{(l)} \leq l$ то
18. $p_{ii} := 1$; $p_{ij} := 1$;
19. иначе $p_{ij} := 0$;
20. конец цикла;
21. $l := l + 1$;
22. конец цикла;
- конец

Последовательное вычисление перархии зон для графа, показанного на рис. 2.21:

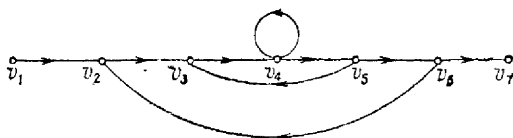


Рис. 2.21.

A	1	2	3	4	5	6	7	D	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0								
2	0	0	1	0	0	0	0	2	0	0	1	0	0	0	0								
3	0	0	0	1	0	0	0	3	0	0	0	1	0	0	0								
4	0	0	0	1	1	0	0	4	0	0	0	1	1	0	0								
5	0	0	1	0	0	1	0	5	0	0	1	0	0	1	0								
6	0	1	0	0	0	0	1	6	0	1	0	0	0	0	1								
7	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0								

A^2	1	2	3	4	5	6	7	D_2	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1	0	0	1	0	0	0	0	1	0	1	2	0	0	0	0								
2	0	0	0	1	0	0	0	2	0	0	1	2	0	0	0								
3	0	0	0	1	1	0	0	3	0	0	0	1	2	0	0								
4	0	0	1	1	1	1	0	4	0	0	2	1	1	2	0								
5	0	1	0	1	0	0	1	5	0	2	1	2	0	1	2								
6	0	0	1	0	0	0	0	6	0	1	2	0	0	0	1								
7	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0								

A^3	1	2	3	4	5	6	7	D_3	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1 0	0	0	1	0	0	0	0	1 0	1	2	3	0	0	0	0	0 0	0	1	0	0	0	0	
2 0	0	0	1	1	0	0	0	2 0	0	1	2	3	0	0	0	0 0	1	1	1	0	0	0	
3 0	0	1	1	1	1	0	0	3 0	0	3	1	2	3	0	0								
4 0	1	1	1	1	1	1	1	4 0	3	2	1	1	2	3									
5 0	0	1	1	1	0	0	0	5 0	2	1	2	3	1	2									
6 0	0	0	1	0	0	0	0	6 0	1	2	3	0	0	1									
7 0	0	0	0	0	0	0	0	7 0	0	0	0	0	0	0									
A^4	1	2	3	4	5	6	7	D_4	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1 0	0	0	1	1	0	0	0	1 0	1	2	3	4	0	0	0	0 0	0	1	0	0	0	0	
2 0	0	1	1	1	1	0	0	2 0	0	1	2	3	4	0	0	0 0	1	1	1	0	0	0	
3 0	1	$\overline{1}$	1	1	1	1	1	3 0	4	3	1	2	3	4									
4 0	1	1	$\overline{1}$	1	1	1	1	4 0	3	2	1	1	2	3									
5 0	0	1	1	$\overline{1}$	1	0	0	5 0	2	1	2	3	1	2									
6 0	0	0	1	1	0	0	0	6 0	1	2	3	4	0	1									
7 0	0	0	0	0	0	0	0	7 0	0	0	0	0	0	0									
A^5	1	2	3	4	5	6	7	D_5	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1 0	0	1	1	1	1	0	0	1 0	1	2	3	4	5	0	0	0 0	0	1	0	0	0	0	
2 0	$\overline{1}$	1	1	1	1	1	1	2 0	5	1	2	3	4	5	0	0 0	1	1	1	0	0	0	
3 0	1	$\overline{1}$	1	1	1	1	1	3 0	4	3	1	2	3	4	0	0 1	1	1	1	1	0	0	
4 0	1	1	$\overline{1}$	1	1	1	1	4 0	3	2	1	1	2	3									
5 0	1	1	1	$\overline{1}$	1	1	1	5 0	2	1	2	3	1	2									
6 0	0	1	1	1	$\overline{1}$	0	0	6 0	1	2	3	4	5	1									
7 0	0	0	0	0	0	0	0	7 0	0	0	0	0	0	0									
A^6	1	2	3	4	5	6	7	D_6	1	2	3	4	5	6	7	P	1	2	3	4	5	6	7
1 0	1	1	1	1	1	1	1	1 0	1	2	3	4	5	6	0 0	0	0	1	0	0	0	0	
2 0	1	1	1	1	1	1	1	2 0	5	1	2	3	4	5	0 0	1	1	1	1	0	0	0	
3 0	1	1	1	1	1	1	1	3 0	4	3	1	2	3	4	0 1	1	1	1	1	1	0	0	
4 0	1	1	1	1	1	1	1	4 0	3	2	1	1	2	3									
5 0	1	1	1	1	1	1	1	5 0	2	1	2	3	1	2									
6 0	1	1	1	1	1	1	1	6 0	1	2	3	4	5	1									
7 0	0	0	0	0	0	0	0	7 0	0	0	0	0	0	0									

Окончательно имеем иерархию вложенных зон:

$B_1: \{v_1\}$; $B_2: \{v_3, v_4, v_5\}$; $B_3: \{v_2, v_3, v_4, v_5, v_6\}$.

2.9. Иерархия зон и прямая нумерация. Для графа с прямой нумерацией вершин легко найти множество вложенных зон, которые в данном случае являются

формальными циклами. Самый внутренний формальный цикл — это такой формальный цикл $R_i = F[v_a : v_b]$, что не существует формального цикла $R_j = F[v_c : v_d]$, для которого $F[v_c : v_d] \subset F[v_a : v_b]$. Самый внутренний формальный цикл есть примитивный формальный цикл, но обратное неверно.

Теорема 6. Для самого внутреннего формального цикла $R = F[v_a : v_b]$ единственной обратной дугой к вершинам в R служит дуга (v_b, v_a) .

Доказательство. Так как самый внутренний формальный цикл есть примитивный формальный цикл, то имеем дугу (v_b, v_a) . Не может быть обратной включающей дуги (v_a, v_c) , где либо $a \leq c \leq d \leq b$, либо $a < c \leq d \leq b$, так как в противном случае $F[v_c : v_d]$, а не $F[v_a : v_b]$ был бы самым внутренним формальным циклом. Также не может быть обратной дуги (v_a, v_c) , где $a < c \leq b < d$, потому что тогда R не был бы формальным циклом.

Определим теперь редуцированный граф как граф, получаемый путем применения к графу с прямым порядком вершин, содержащему формальный цикл, преобразований:

1. Выбрать самый внутренний формальный цикл $R = F[v_a : v_b]$.

2. Удалить все вершины R из графа и заменить их одной вершиной v_R , такой, что:

а) $F(v_R) = a$;

б) дуга (v_j, v_R) существует тогда и только тогда, когда $v_j \notin R$ и была некоторая вершина $v_k \in R$, для которой существовала дуга (v_j, v_k) (заметим, что $j < a$, так как R — формальный цикл);

в) дуга (v_R, v_m) существует тогда и только тогда, когда $v_m \notin R$ и была некоторая вершина $v_n \in R$, для которой существовала дуга (v_n, v_m) .

Пример редукции графа показан на рис. 2.22.

Теорема 7. Редукция графа сохраняет его прямой порядок.

Доказательство. Ясно, что все пути вне R , которые существовали до редукции, продолжают существовать и после нее. Исключением являются те, в которые не была включена последовательность вершин из R . Вместо нее после редукции будет единственная вершина v_R . Более того, все новые дуги будут прямыми или обратными в соответствии с тем, были замененные ими дуги прямыми или обратными. Таким образом,

редукция сохраняет все три свойства, характеризующие прямой порядок.

Теорема 8. Для графа с любой прямой нумерацией вершин F всякая зона есть подмножество формального цикла и включает его начальную вершину.

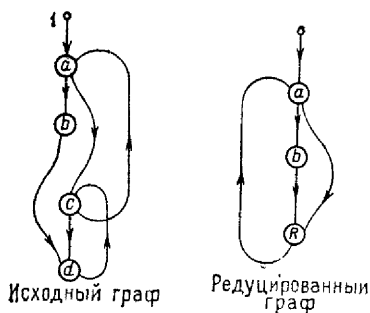


Рис. 2.22.

Доказательство. Предположим, что $R = \{v_{a_1}, \dots, v_{a_k}\}$ есть множество вершин некоторой зоны, причем $F(v_{a_1}) < F(v_{a_2}) < \dots < F(v_{a_k})$. По определению зоны внутри R существует путь из v_{a_k} в v_{a_1} . Такой путь включает обратную дугу в

v_{a_1} (прямая дуга в v_{a_1} будет обязательно из вершины вне зоны). Из определения формального цикла ясно, что v_{a_1} есть начальная вершина одного или более формальных циклов. Выберем из них один, имеющий наибольшее число вершин. Пусть это будет C . Предположим, что найдется вершина v_r в R , не принадлежащая C . Так как вершины в C занумерованы последовательно и $v_{a_1} \in C$, а также $F(v_r) > F(v_{a_1})$, то $F(v_r) > F(v_i)$ для любой вершины $v_i \in C$. По определению зоны существует путь из v_r в v_{a_1} , содержащий обратную дугу из вершины вне C в вершину C . Но тогда существует формальный цикл с начальной вершиной v_{a_1} , который содержит больше вершин, чем C , что противоречит предположению. Итак, каждая вершина из R одновременно принадлежит и C .

Из теоремы 8 следует, что для каждой обратной дуги найдется последовательно занумерованное множество вершин, порождающее зону. Первая вершина этой зоны есть начальная вершина обратной дуги; последняя вершина есть конец самой последней обратной дуги, которая сцеплена с первой (см. п. 1.7). Множество определенных таким образом рангов есть множество вложенных зон.

Самый внутренний формальный цикл есть множество вершин под одной обратной дугой. Если его заменить отдельной вершиной, полученный редуцированный

граф будет снова иметь прямой порядок. Повторяя процедуру, получаем последовательность вложенных зон.

2.10. Иерархия зон и K -нумерация. Пусть построены K -нумерации для $i \in [1:n]$. Обозначим через D множество всех нетривиальных подграфов, принадлежащих множеству $\{K_i[i] \mid i \in [1:n]\}$.

Теорема 9. D — иерархия зон графа G .

Докажем несколько вспомогательных утверждений. Непосредственно из леммы 1 следует

Лемма 10. Для любой вершины $v \in V$ и любой вершины q из $N[v]$ следующие свойства эквивалентны:

- 1) q — входная вершина подграфа $N[v]$;
- 2) q — начальная вершина подграфа $N[v]$;
- 3) существует предшественник w вершины q , для которого $N(w) < N(v)$.

Лемма 11. Для любого $i \in [1:n]$ и любой вершины p имеет место равенство $K_i[p] = N[p]$.

Доказательство проведем индукцией по i . При $i = 1$ утверждение очевидно. Предположим, что лемма верна для всех $i < s$, и рассмотрим случай $i = s$. Если p — некоторая вершина графа, то либо $K_{s-1}(p) < s - 1$, либо $K_{s-1}(p) \geq s - 1$.

Пусть $K_{s-1}(p) < s - 1$. Из определения K -нумерации получаем $K_{s-1}[K_{s-1}(p):n] = K_s[K_s(p):n]$. Это значит, что $K_s[s] = K_{s-1}[p]$ в силу определения F -области и по предположению индукции $K_s[p] = N[p]$.

Пусть теперь $K_{s-1}(p) \geq s - 1$. Нетрудно увидеть, что либо $p \notin K_{s-1}[s - 1]$, либо $p \in K_{s-1}[s - 1]$. Рассмотрим первый случай. Здесь пересечение $K_{s-1}[s - 1] \cap K_{s-1}[p]$ пусто в силу индуктивного предположения и следствия 2 из леммы 9. Таким образом, по определению K -нумерации $K_{s-1}[p] \subseteq K_s[K_s(p):n] \subseteq K_{s-1}[K_{s-1}(p):n]$, и, следовательно, $K_s[p] = K_{s-1}[p] = N[p]$. Рассмотрим теперь случай $p \in K_{s-1}[s - 1]$. Из определения K -нумерации получаем, что $K_{s-1}[K_{s-1}(p):n] \subseteq K_s[K_s(p):n]$, и если вершина p достижима из некоторой вершины q в $K_s[K_s(p):n]$, то $q \in K_{s-1}[K_{s-1}(p):n]$. Таким образом, $K_s[p] = K_{s-1}[p] = N[p]$.

Следствие. Множество $\{K_i[i] \mid i \in [1:n]\}$ совпадает с множеством $\{N[p] \mid p \in X\}$.

Доказательство теоремы 9. Если вершина p такова, что $N(p) = \min \{N(q) \mid q \in S\}$, где S — произвольная зона, то $S \subseteq N[p]$. Отсюда по лемме 9, по следствию 2 из нее, следствию из леммы 11 и по лемме 10 получаем, что D — иерархия зон графа G .

2.11. Отыскание линейных компонент. Бивершину v назовем *точкой сочленения* графа G , если ее T -номер $T(v)$ совпадает с наибольшим T -номером приемников вершин из $T[1 : T(v) - 1]$.

Теорема 10. Пусть $\{v_1, \dots, v_{k+1}\}$ — множество всех точек сочленения графа G , упорядоченных в порядке возрастания T -номеров. Подграф H графа G есть линейная компонента с начальной вершиной p и конечной вершиной q тогда и только тогда, когда $H = T[T(v_j) : T(v_{j+1}) - 1]$, $p = v_j$, $q = v_{j+1}$, для некоторого $j \in [1 : k]$.

Доказательство. Очевидно, что теорема верна, если граф является бесконтурным, поскольку в этом случае согласно теореме 4 и определению T -нумерации для любой дуги (p, q) графа имеем неравенство $T(p) < T(q)$.

Рассмотрим общий случай. Заметим, что если линейной компоненте H принадлежит некоторая вершина бикомпоненты графа, то H содержит все вершины этой бикомпоненты. Если p — либо начальная, либо конечная вершина линейной компоненты и принадлежит некоторой бикомпоненте графа, то p — единственная входная вершина этой бикомпоненты.

Пусть G^* есть граф Герца графа G . Подграф H графа G является его линейной компонентой с начальной вершиной p и конечной q тогда и только тогда, когда существует последовательность H_1, H_2, \dots, H_s линейных компонент графа G^* , для которой выполнены условия: а) подграф $\cup \{H_i \mid i \in [1 : s]\}$ графа G^* соответствует подграфу H графа G ; б) для каждого i из $[1 : s - 1]$ начальная вершина у H_{i+1} — это конечная вершина у H_i , соответствующая бикомпоненте G , которая имеет более одной входной вершины; в) начальная вершина у H_1 и конечная у H_s соответствуют одновходовым бикомпонентам S_1 и S_2 графа G , а p и q — входные вершины соответственно у S_1 и S_2 . Таким образом, по определению линейной компоненты, согласно лемме 10 и теореме 4 общий случай сводится к случаю, когда G бесконтурный. Теорема доказана.

Итак, для нахождения линейных компонент необходимо найти N -нумерацию, затем T -нумерацию вершин графа и множество BV его бивершин. Выделение линейных компонент происходит при обработке вершин графа в порядке возрастания T -номеров.

А л г о р и т м:

В х о д: Граф $G = (V, E)$, заданный списками смежности $A(v)$, T -нумерация вершин, множество BV бивершин.

В ы х о д: Множество линейных компонент.

начало

1. взять в качестве u вход s графа G ;
 2. $N1 := 1$;
 3. для k от 1 до $n - 1$ шаг 1 цикл
 4. **начало**
 5. выбрать вершину v , у которой $T(v)$ равен k ;
 6. для всех w из $A(v)$ цикл
 7. если $N1 < T(w)$ то $N1 := T(w)$;
 8. **конец цикла**;
 9. выбрать вершину z , у которой $T(z)$ равен $k + 1$;
 10. если $N1 = k + 1 \wedge z \in BV$ то
 11. занести $T[u : T(z) - 1]$ в список линейных компонент как линейную компоненту с начальной вершиной u и конечной z ;
 12. взять в качестве u вершину z ;
 13. **конец цикла**;
- конец**

Алгоритм вычисляет наибольший T -номер преемников вершин из $T[1 : T(p) - 1]$ для каждой вершины p при обработке вершин в порядке возрастания T -номеров и правильно выделяет линейные компоненты графа согласно теореме 10. Трудоемкость равна $O(m)$, так как каждая дуга обрабатывается в точности один раз.

2.12. Для неориентированных графов задачей, эквивалентной задаче о нахождении линейных компонент, является задача о нахождении блоков (задача об отыскании двусвязных компонент). Разделению графа на блоки соответствует разбиение множества его ребер на непересекающиеся классы по отношению эквивалентности R : $R(u_1, u_2) \Leftrightarrow$ существует простой цикл, содержащий ребра u_1 и u_2 .

Ветвью блоков, висящей на разделяющей вершине z , называется подграф $G' = \langle V' \cup \{z\} \rangle$, $V' \ni z$, такой, что вершины из V' не соединены ребрами с вершинами из $V \setminus (V' \cup \{z\})$ и подграф $\langle V' \rangle$ связан. Очевидно, что вершина z принадлежит только одному блоку в графе G' , который называется *корневым* для вет-

ви G' . Пусть фиксирована вершина a . Каждый блок B (кроме $B \ni a$, если a — не разделяющая в G) выступает как корневой для единственной ветви блоков из класса $\{G' | a \notin V'\}$.

Нахождение блоков в графе осуществляется алгоритмом, основанным на поиске в глубину, с трудоемкостью $O(m)$, где m — число ребер, если граф задан списками смежности.

Назовем обход графа, соответствующий поиску в глубину, *стандартным* обходом; траекторию обхода графа будем обозначать Q , начало обхода — буквой a . Начальная вершина выбирается произвольно. Будем считать, что M -номера вершин соответствуют последовательным моментам обхода, а ребра маршрута — Q -ходам обхода. Первый ход по ребру, во время которого его конец получает M -номер, назовем *прямым ходом*, второй — в противоположном направлении — *обратным*. Условимся, что прямой ход задает ориентацию ребер, связанную с данным обходом. Запись $u = (y, z)$ будет означать, что ребро ориентировано от y к z . Ребро, по которому мы впервые попадаем в вершину v , обозначим через u_v . Основное свойство стандартного обхода в данном случае состоит в том, что, войдя в ветвь блоков G' по ребру u , мы находимся внутри G' до тех пор, пока не обойдем ее полностью, заканчивая движение по G' обратным ходом по u . Для удобства изложения будем называть момент обхода, при котором мы прямым ходом попадаем в вершину, уже фигурировавшую в обходе (т. е. следующий ход должен быть обратным по тому же ребру), *касательным моментом*. Соответствующее ребро назовем *касательным*.

Рассмотрим часть \bar{Q} обхода Q , получаемую при выбрасывании из Q всех касательных ребер. \bar{Q} может рассматриваться как обход части \bar{G} графа G , образованной некасательными ребрами, т. е. ребрами типа u_v , где v пробегает множество $V \setminus \{a\}$. Легко проверить, что \bar{G} — дерево, ориентированное от корня a . Для каждой вершины существует единственный путь $L_x \subset \bar{G}$ из a в x .

Будем называть ребро, не фигурировавшее в обходе, *непройденным*, пройденное один раз — *полупройденным*, пройденное два раза — *пройденным*. По индукции можно доказать, что множество полупройденных ребер в любой некасательный момент α обхода Q

совпадает с множеством ребер «текущего» пути $L_\alpha = \equiv L_{v_\alpha}$.

Назовем вершину *свободной*, если все инцидентные ей ребра непройденные, *замкнутой*, если все они пройденные, и *открытой* — в остальных случаях. Рассмотрим вершину $v \neq a$. Очевидно, что она перестает быть свободной в тот момент, когда включается в текущий путь вместе с входящим в него ребром u_v . Докажем, что вершина v становится замкнутой в тот момент, когда текущий путь перестает включать v и u_v в результате обратного хода по ребру u_v . Действительно, в этом случае вершине v не инцидентны непройденные ребра, так же как и полупройденные, что и требовалось доказать. Аналогично доказывается, что в момент ω окончания обхода вершина a замкнута. Тем самым доказано, что множество открытых вершин совпадает с множеством вершин текущего пути, не считая начального и конечного моментов, когда оно пусто. Итак, в момент окончания обхода все ребра пройдены и все вершины замкнуты.

Из свойств поиска в глубину известно, что дерево \bar{G} представляет собой каркас графа G .

Л е м м а 12. Пусть $u = (y, z) \in U$ — хорда каркаса \bar{G} , т. е. $u \notin \bar{G}$. Тогда $z \in L_v$.

Д о к а з а т е л ь с т в о. По определению \bar{G} ребро u — касательное. Пусть прямой ход по нему был совершен в момент α , т. е. $v_\alpha = y$ и $v_{\alpha+1} = z$. Вершина z не свободна по определению касательности и не замкнута по определению замкнутости. Значит, z открыта и $z \in L_\alpha = L_v$.

По структуре, задаваемой на графе G каркасом \bar{G} , однозначно определяется любая ветвь блоков \bar{G} , $V' \neq a$, а именно множество вершин ветви блоков образует в каркасе \bar{G} отделимую ветвь. Для уточнения этого понятия рассмотрим часть каркаса \bar{G} , отделяемую ребром u от корня дерева \bar{G} — вершины a . Присоединим к ней само ребро $u = (z, y)$. Полученную часть \bar{G}_u каркаса \bar{G} будем называть ветвью, висящей на ребре u и на вершине z . Множество вершин ветви \bar{G}_u , кроме z , обозначим V_u . Ветвь \bar{G}_u будем называть *отделимой*, если из вершин множества V_u не исходит ребер в вершины из $V \setminus (V_u \cup \{z\})$, точнее, в вершины пути \bar{L}_z (кроме z). Очевидно, что отделимая ветвь \bar{G}_u , $u = (z, y)$, задает ветвь блоков с множеством внутренних вершин $V_u \neq a$, висящую на вершине z .

Рассмотрим ветвь блоков \bar{G}_u , $u = (z, y)$, висящую на вершине z . Так как $\bar{G} \subset G$, вершина z в каркасе \bar{G} отделяет V' от остальных вершин. Значит, V' образует в \bar{G} одну или несколько ветвей, висящих на z . Однако в силу леммы 12 внутренние вершины из различных ветвей, висящих на одной вершине, не могут соединяться ребрами. Поэтому связность подграфа $\langle V' \rangle$ влечет за собой единственность такой ветви.

Обратимся к задаче нахождения отделимых ветвей в процессе обхода. Очевидно, обход ветви \bar{G}_u есть часть обхода Q , начинающаяся прямым и кончающаяся обратным ходом по u . Часть Q , заключенную в этом промежутке, назовем ветвью обхода и обозначим Q_u . Определение отделимости распространим на ветви обхода: Q_u отделима $\Leftrightarrow \bar{G}_u$ отделима. Из свойств обхода вытекает, что ветвь обхода Q_u при $u = (z, y)$ представляет собой обход подграфа $G_u = \langle V_u \cup \{z\} \rangle$ плюс касания с вершинами пути L_z (кроме z). Тем самым можно считать обоснованным *критерий отделимости*:

Ветвь Q_u , где $u = (z, y)$, отделима в том и только том случае, когда на протяжении Q_u не было касаний с вершинами текущего пути, расположенными на нем ближе к началу (вершине a), чем вершина z .

Покажем, как можно использовать этот критерий в процессе обхода. Каждое некасательное ребро с момента его появления в текущем пути, т. е. после прямого хода по нему, будем считать «подозрительным»; метка подозрительности снимается позже. Пусть в момент α был совершен прямой ход по касательному ребру (y, z) , $v_\alpha = y$, $v_{\alpha+1} = z \in L_\alpha$. Тогда все ребра контура $L_{\alpha+1}$, кроме ребра, исходящего из z , не могут порождать отделимую ветвь обхода. Поэтому все такие ребра, имеющие в рассматриваемый момент метку подозрительности, освобождаются от нее. Очевидно, что отделимость ветви обхода Q_u эквивалентна тому, что к моменту обратного хода по ребру u оно еще считается подозрительным.

Таким образом, получен алгоритм нахождения отделимых ветвей обхода. Необходимо только указать способ стирания меток подозрительности, не требующий в каждый момент касания просмотра всех ребер образующегося контура. Для этого нужно иметь список «подозрительных» ребер (с. п. р.) в том порядке, в каком они расположены в текущем пути, и аккуратно организовать вычеркивание элементов из с. п. р. Дело

в том, что в момент касания из с. п. р. выбрасывается «хвост», и задача состоит в отыскании элемента, который становится последним. Этот элемент можно было бы найти просмотром с. п. р. с конца, если бы вершины обладали признаком, позволяющим определить порядок их расположения в текущем пути. Для этого вершины графа нужно нумеровать в процессе обхода (M -нумерация). Текущий путь изменяется только с «хвоста», причем при наращивании к нему присоединяется вершина с номером большим, чем любой ранее присвоенный. Поэтому для любого α на пути L_α номера вершин возрастают от a к v_α .

Покажем теперь, как нахождение отдельных ветвей обхода влечет за собой выделение блоков графа G . Пусть в момент обратного хода по ребру $u = (z, y)$ обнаружилось, что ветвь Q_u отделима. Тогда множество внутренних вершин соответствующей ветви блоков есть V_u , и оно может быть найдено как множество вершин, встретившихся в обходе не ранее, чем вершина y . Для того чтобы в момент нахождения ветви блоков выделить множество вершин корневого блока, достаточно при нахождении каждой ветви блоков множество ее внутренних вершин исключать из дальнейшего рассмотрения. Тогда к завершению обхода любой ветви блоков G' все подветви, висящие на корневом блоке B , будут обойдены, а их внутренние вершины исключены. Останутся лишь вершины из B . Для экономного выполнения операций предлагается вести текущий список вершин (т. с. в.) в порядке возрастания номеров. Исключение вершин из т. с. в. будет сводиться к отбрасыванию его «хвоста».

Алгоритм выделения блоков:

Шаг 1. Пусть в момент α мы попадаем впервые в вершину v . Присваиваем ей очередной M -номер $M(v)$ и заносим в конец массива т. с. в. Ребро u_v заносим в конец массива с. п. р.

Шаг 2. Пусть был сделан прямой ход по касательному ребру $(v_\alpha, v_{\alpha+1})$. Просматриваем с. п. р. с конца до первого его элемента — ребра (x, y) , для которого $M(x) \leq M(v_{\alpha+1})$. Это ребро считаем далее последним в с. п. р.

Шаг 3. Пусть обратным ходом по ребру $u_y = (z, y)$ «возвращаемся» в вершину z . Если ребро u_y стоит последним в с. п. р., то переносим конец с. п. р. на ячейку назад, помечаем вершину z как разделяющую

(если этого признака у нее еще нет) и заносим z в список нового блока B . Просматриваем т. с. в. с конца, занося просмотренные вершины в список вершин блока B . Процесс продолжаем до тех пор, пока не дойдем до вершины y и не занесем ее. Далее, последней в т. с. в. будем считать предыдущую для y вершину.

Шаг 4. В конце обхода все вершины из т. с. в. заносим в список очередного блока A ($a \in A$), если в т. с. в. кроме a содержится хотя бы одна вершина.

§ 3. Гамаки, полугамаки и шлейфы

3.1. Свойства M - и L -нумераций.

Лемма 13. Для любого $i \in [0 : n]$ и любой вершины $p \in X$ из условия $L_i(p) \leq i$ следует

$$L_i[p] = N[p] = L_i[L_i(p) : L_i(p) + |L_i(p)| - 1].$$

Доказательство проведем индукцией по i . Для $i = n$ утверждение леммы непосредственно следует из определений K - и L -нумераций, леммы 11 и следствия 2 из леммы 9. Предположим, что лемма справедлива для всех $i > k$, и рассмотрим случай, когда $i = k$. Пусть p — такая вершина графа, что $L_k(p) = j \leq k$. Тогда по определению L -нумерации получаем, что $L_{k+1}(p) = j$ и $L_k[j : n] = L_{k+1}[j : n]$. Это означает, что $L_k[j] = L_{k+1}[j]$. Однако так как $L_k(p) = j \leq k$, то по индуктивному предположению и следствию 2 из леммы 9 получаем

$$L_{k+1}[j] = L_{k+1}[j : j + |L_{k+1}[p]| - 1] = N[p]$$

и либо $L_{k+1}[k+1] \subseteq L_{k+1}[j]$, либо $L_{k+1}[k+1] \cap L_{k+1}[j] = \emptyset$.

Следовательно, по определению L -нумерации

$$L_{k+1}[j] = L_k[j : j + |L_{k+1}[p]| - 1] = N[p].$$

Это завершает доказательство, поскольку $L_k[j] = L_{k+1}[j]$.

Будем обозначать через H гамак с начальной вершиной p и конечной q , а через \bar{H} — множество $H \cup \{q\}$.

Лемма 14. Для любого $i \in [1 : n]$ имеет место одна из трех альтернатив: $N[i] \subseteq H$; $\bar{H} \subset N[i]$; $H \cap N[i] = \emptyset$.

Доказательство. Нетрудно проверить, что если S — некоторая зона графа G , то либо $S \subseteq H$, либо $\bar{H} \subset S$, либо $S \cap H = \emptyset$. Отсюда по лемме 9 получаем

нужное утверждение, поскольку согласно определению гамака $N(p) = \min \{N(p_k) | p_k \in H\}$.

Л е м м а 15. Пусть $\mu_1 = [p_1, \dots, p_k]$ и $\mu_2 = [q_1, \dots, q_s]$ — такие M -пути, что $M(p_1) \leq M(q_1) \leq M(p_k) \leq M(q_s)$. Тогда существует M -путь из q_1 в p_k .

Д о к а з а т е л ь с т в о. Предположим, что не существует M -пути из вершины q_1 в вершину p_k . Тогда справедливы неравенства $M(p_1) < M(q_1) < M(p_k)$. Из M -достижимости вершин в орграфе с одним входом получаем, что $M(q_1) < M(v)$ и v принадлежит M -путям из входа s до вершин из μ_1 . Пусть u и z — такие вершины графа, что $M(u) = \min \{M(v) | v \in A\}$ и $M(z) = \max \{M(v) | M(v) < M(u), v \in B(u)\}$. Тогда $M(z) < M(q_1) < M(u)$, $z \in B(u)$ и множество $\hat{B} = M[M(z) + 1 : M(u) - 1]$ не содержит предшественников вершины u . Отсюда по определению M -нумерации получаем, что все преемники вершин из \hat{B} имеют M -номера меньше, чем M -номер вершины u . Но вместе с тем $q_1 \in \hat{B}$, $M(u) < M(q_s)$ и μ_2 есть M -путь из q_1 в q_s . Следовательно, некоторый преемник вершин из \hat{B} имеет M -номер не меньший, чем $M(u)$. Таким образом, имеет место противоречие.

Л е м м а 16. Если $M(p) < M(q)$, то

$$N[N(p) : N(q)] \subseteq \bar{H} \subseteq N[N(p) : n].$$

Д о к а з а т е л ь с т в о. Из определения гамака и следствия из леммы 1 получаем, что $M(p) = \min \{M(u) | u \in H\}$. Согласно лемме 1 и определению N -нумерации отсюда следует, что $\bar{H} \subseteq N[N(p) : n]$. Осталось показать, что $N[N(p) : N(q)] \subseteq \bar{H}$. В силу определения N -нумерации получаем, что $p_1 \in N[N(p) + 1 : N(q) - 1]$ и $M(p) < M(q)$ тогда и только тогда, когда выполняется одно из следующих условий:

- 1) $M(p) < M(p_1) < M(q)$ и имеются M -пути из p в p_1 и из p_1 в q ;
- 2) $M(p_1) < M(p) < M(q)$, имеется M -путь из p_1 в q и нет M -пути из p_1 в p ;
- 3) $M(p) < M(q) < M(p_1)$, имеется M -путь из p в p_1 и нет M -пути из q в p_1 .

При выполнении условий 1) или 3) в силу определения гамака имеет место включение $p_1 \in \bar{H}$. Свойство 2) не может иметь места согласно лемме 15 и следствию из леммы 1.

Л е м м а 17. Если $M(p) > M(q)$, то $H = N[N(p) : N(p) + |H| - 1]$.

Так же как и в доказательстве предыдущей леммы, имеем $M(p) = \min \{M(u) \mid u \in H\}$. Рассмотрим две такие вершины v и w , что v принадлежит гамаку H , а $M(p) < M(w) < M(v)$. Применяя леммы 1 и 15, получаем существование M -пути из p в w . Ясно, что вершина q не может принадлежать этому пути, поскольку $M(q) < M(p)$, и, следовательно, по определению гамака $w \in H$. Таким образом, $H = M[M(p) : M(p) + |H| - 1]$ и к любой вершине из H существует M -путь из вершины p . Теперь утверждение леммы вытекает из определения гамака и N -нумерации.

Лемма 18. Если $M(q) < M(p)$, то для любого $i \in [0 : n]$ имеем $H = L_i[L_i(p) : L_i(p) + |H| - 1]$.

Доказательство проведем индукцией по i . Пусть $i = n$. Если для некоторого $j \in [1 : n - 1]$ выполняется равенство $H = K_j[K_j(p) : K_j(p) + |H| - 1]$, то в силу лемм 11, 14 и определения K -нумерации выполняется равенство $H = K_{j+1}[K_{j+1}(p) : K_{j+1}(p) + |H| - 1]$. Отсюда следует справедливость леммы для случая $i = n$, поскольку согласно лемме 17 имеет место равенство $H = N[N(p) : N(p) + |H| - 1]$.

Предположим, что лемма верна для всех $i \in [s : n]$, и рассмотрим случай, когда $i = s - 1$. Из лемм 13 и 14 следует, что либо $\bar{H} \subset L_s[s]$, либо $L_s[s] \subseteq H$, либо $H \cap L_s[s] = \emptyset$. Таким образом, достаточно рассмотреть случай, когда $\bar{H} \subset L_s[s]$. В этом случае либо все вершины из H имеют один и тот же L_s -ранг, либо L_s -ранг любой вершины из H отличается от L_s -ранга любой вершины из $L_s[s] \setminus H$ и равен $L_s(w)$ для некоторой вершины $w \in H$. Это означает, что $H = L_{s-1}[L_{s-1}(p) : L_{s-1}(p) + |H| - 1]$.

Лемма 19. Если $M(p) < M(q)$, то $\bar{H} = L[p : q]$.

Доказательство. Индукцией по i покажем, что справедливо более общее утверждение для случая, когда $M(p) < M(q)$: для любого $i \in [0 : n]$, если $i < L_i(p)$, то $\bar{H} = L[p : q]$; в противном случае $L_i[p : q] \subseteq \bar{H} \subseteq L_i[L_i(p) : n]$.

Пусть $M(p) < M(q)$. Рассмотрим случай, когда $i = n$. По лемме 16 имеем $N[p : q] \subseteq \bar{H} \subseteq N[N(p) : n]$. Из лемм 11, 14 и определения K -нумерации следует, что если $K_j[p : q] \subseteq \bar{H} \subseteq K_j[K_j(p) : n]$ для некоторого $j \in [1 : n - 1]$, то $K_{j+1}[p : q] \subseteq \bar{H} \subseteq K_{j+1}[K_{j+1}(p) : n]$. Таким образом, лемма справедлива при $i = n$. Пусть утверждение верно для всех $i \geq s$ и пусть $i = s - 1$. Рассмотрим два случая: $L_{s-1}(p) > s - 1$ и $L_{s-1}(p) \leq s - 1$.

Пусть $L_{s-1}(p) > s - 1$. Тогда по определению L -нумерации $L_s(p) \geq s$. Предположим, что $L_s(p) > s$. Тогда по индуктивному предположению $\bar{H} = L_s[p : q]$. Отсюда и из лемм 13 и 14 получаем, что достаточно рассмотреть случай, когда $\bar{H} \subseteq L_s[s]$. Но в этом случае либо все вершины из \bar{H} имеют один и тот же L_s -ранг, либо L_s -ранг каждой вершины $v \in \bar{H}$ совпадает с $L_s(w)$ для некоторой $w \in \bar{H}$ и отличается от L_s -ранга любой вершины из $L_s[s] \setminus \bar{H}$, если $v \neq q$. Таким образом, $\bar{H} = L_{s-1}[p : q]$. Пусть $L_s(p) = s$. Тогда по предположению индукции $L_s[p : q] \subseteq \bar{H} \subseteq L_s[L_s(p) : n]$ и согласно леммам 13 и 14 $\bar{H} \subseteq L_s[p]$. Отсюда и из определений гамма-ка и F -ранга получаем, что L_s -ранг любой вершины $v \in H$ меньше L_s -ранга q и, если L_s -ранг некоторой вершины $v \in L_s[p]$ меньше L_s -ранга вершины q , то $v \in \bar{H}$. Таким образом, в силу определения L -нумерации получаем $\bar{H} = L_{s-1}[p : q]$.

Пусть $L_{s-1}(p) \leq s - 1$. Тогда по определению L -нумерации и предположению индукции $L_s(p) < s$ и $L_s[p : q] \subseteq \bar{H} \subseteq L_s[L_s(p) : n]$. Из неравенства $L_s(p) < s$ согласно леммам 13 и 14, а также по определению L -нумерации получаем, что либо $L_s(v) = L_{s-1}(v)$ для любой вершины $v \in \bar{H}$, либо для любой вершины $v \in H$ выполнено следующее свойство: если $L_s(v) \neq L_{s-1}(v)$, то $v \in H \setminus \{p\}$ и существует такая вершина w из H , отличная от p , что $L_s(v) = L_{s-1}(w)$. Таким образом, $L_{s-1}[p : q] \subseteq \bar{H} \subseteq L_{s-1}[L_{s-1}(p) : n]$. Лемма доказана.

Пусть S — некоторый подграф графа G и p — некоторая вершина из S . Определим множества и величины:

$$\mathfrak{M}_1(S, p) = \{v \in A(w) \mid w \in S \text{ \& } M(v) < M(p)\};$$

$$\mathfrak{M}_2(S, p) = \{v \in B(w) \mid w \in S \text{ \& } w \neq p \text{ \& } L(v) < L(p)\};$$

$$\mathfrak{M}_3(S, p) =$$

$$= \{v \in A(w) \mid w \in S \text{ \& } M(v) > M(p) \text{ \& } L(v) < L(p)\};$$

$$m_1^*(S, p) = \max(L(p), \{L(v) \mid v \in A(w) \text{ \& } w \in S \text{ \& } M(v) > M(p)\});$$

$$m_2^*(S, p) = \max(L(p), \{L(v) \mid v \in B(w) \text{ \& } w \in S \text{ \& } w \neq p\}).$$

Теорема 11. Подграф H графа G является гамма-ком тогда и только тогда, когда выполнены следующие условия:

1) $H = L[L(p) : L(p) + |H| - 1]$;

2) $M(v) > M(p)$ для любой вершины v из H ;

3) $\mathfrak{M}_2(H, p) = \emptyset$;

4) $\mathfrak{M}_3(H, p) = \emptyset$;

5) $p \notin A(q)$;

6) $m_2^*(H, p) < |H| + L(p)$;

7) либо $\mathfrak{M}_1(H, p) = \emptyset$ и $m_1^*(H, p) = L(p) + |H| = L(q)$, либо $\mathfrak{M}_1(H, p) = \{q\}$ и $m_1^*(H, p) < L(p) + |H|$.

Доказательство. Для любого гамака указанные условия выполняются в силу определения гамака и лемм 18, 19, а также следствия из леммы 1.

Пусть некоторый подграф H удовлетворяет условиям 1)–7). Из условий 3) и 6) следует, что H имеет единственную начальную вершину p . Условие 7) означает, что H имеет конечную вершину q , для которой либо $L(q) = L(p) + |H|$, если множество $\mathfrak{M}_1(H, p)$ пусто, либо $q \in \mathfrak{M}_1(H, p)$. Предположим, что у H существует конечная вершина v , отличная от q . Из условия 7) получаем, что $M(v) > M(p)$, $L(v) < L(p)$, и, следовательно, $v \in \mathfrak{M}_2(H, p)$. Но $\mathfrak{M}_2(H, p)$ пусто по условию 3). Следовательно, имеет место противоречие. Таким образом, H — гамак с начальной вершиной p и конечной q .

Следствие. Пусть p — некоторая вершина графа и S — подграф $L[L(p) : L(p) + k]$, $k \geq 0$, для которых выполнено по крайней мере одно из следующих условий:

1) $\mathfrak{M}_2(S, p) \neq \emptyset$;

2) $\mathfrak{M}_3(S, p) \neq \emptyset$;

3) в S имеется такая вершина v , что $M(v) < M(p)$;

4) $L(p) + k = n - 1$.

Тогда не существует такого гамака H с начальной вершиной p , что $S \subset H$.

3.2. Алгоритм отыскания гамаков. Отыскание гамаков происходит в результате исполнения $n - 1$ шагов, на каждом из которых находятся все гамаки, имеющие в качестве начальной некоторую фиксированную вершину p . Для этого вначале помечаются все предшественники p , затем последовательно рассматриваются множества подряд занумерованных вершин $L[L(p) : L(p) + i]$, начиная с множества, для которого $i = 0$. Чтобы определить, образует ли текущее множество гамаков, для p и для данного множества вершин вычисляются множества \mathfrak{M}_1 , \mathfrak{M}_2 , \mathfrak{M}_3 и величины m_1^* и m_2^* . После

указанных вычислений происходит проверка условий теоремы 11 и следствия из нее. В зависимости от результатов этой проверки либо осуществляется переход к рассмотрению следующего множества, либо заканчивается нахождение гамаков с данной начальной вершиной.

А л г о р и т м:

В х о д: Граф $G = (V, E)$, заданный списками смежности $A(v)$ и $B(v)$, M -нумерация вершин и соответствующая ей L -нумерация.

В ы х о д: Множество гамаков.

начало

1. процедура ГАМАКИ (M, L, p);
2. **начало**
3. $МАКС1 := МАКС2 := L(p)$; $МНОЖ1 := \text{истина}$;
4. взять в качестве q вершину p ;
5. $H := \emptyset$;
6. пометить каждую вершину v из $B(p)$;
7. для k от $L(p)$ до $n - 1$ шаг 1 цикл
8. **начало**
9. добавить к H вершину v , у которой $L(v) = k$;
10. для всех w из $A(v)$ цикл
11. **начало**
12. если $M(w) < M(p)$ то
13. если $МНОЖ1 \vee w = q$ то
14. $МНОЖ1 := \text{ложь}$;
15. взять в качестве q вершину w ;
16. иначе завершить нахождение гамаков, у которых p — начальная вершина;
17. иначе если $L(w) > L(p)$ то
18. $МАКС1 := \max \{МАКС1, L(w)\}$;
19. иначе завершить нахождение гамаков, у которых p — начальная вершина;
20. **конец цикла**;
21. если v не есть p то
22. для всех w из $B(v)$ цикл
23. **начало**
24. если $L(w) < L(p)$ то
25. завершить нахождение гамаков, у которых p — начальная вершина;
26. иначе $МАКС2 := \max \{МАКС2, L(w)\}$;
27. **конец цикла**;
28. если $\neg МНОЖ1$ то
29. если $МАКС1 < k + 1 \wedge q$ не помечена то

30. H — гамак с начальной вершиной p и конечной q ;

31. завершить нахождение гамаков, у которых p — начальная вершина;

32. иначе выбрать вершину u , у которой $L(u) = k + 1$;

33. если $\text{MAKCI} = k + 1 \wedge u$ не помечена то

34. H — гамак с начальной вершиной p и конечной вершиной q ;

35. иначе;

36. конец цикла;

37. конец процедуры;

38. взять в качестве p вход s графа;

39. ГАМАКИ (M, L, s);

40. для всех v , являющихся конечными вершинами гамаков и отличных от выхода графа, цикл

41. ГАМАКИ (M, L, v);

конец

3.3. Полугамаки и шлейфы. Назовем вершину b *обязательным преемником* вершины a в графе G , если $b \neq a$, a не является конечной вершиной и b принадлежит каждому пути от a до конечной вершины G . Рассмотрим множество $T(a)$ всех обязательных преемников вершины, отличной от входной вершины a . Из определения ясно, что $T(a)$ линейно упорядочено по отношению к свойству быть обязательным преемником и, кроме того, множество $T(a)$ не пусто (ему принадлежит, например, выход графа). Минимальный элемент $T(a)$ по отношению к рассматриваемому свойству называется *фокусом* вершины a в G и обозначается $F(a, G)$. Иными словами, фокус $F(a, G)$ — это ближайший обязательный преемник вершины a при движении от a к выходу графа.

Полугамаком $S(a, G)$ вершины a называется подграф, порожденный всеми вершинами таких путей из a в $F(a, G)$, которые не содержат повторных вхождений вершины $F(a, G)$ (повторные вхождения a допускаются).

Шлейфом вершины a ($a \neq s$) называется полугамак $S(a, G)$, из которого удалена вершина $F(a, G)$; дуги, ранее направленные в $F(a, G)$, становятся свободными. С точки зрения теории графов шлейф не является графом, так как в последнем не допускаются свободные дуги, т. е. дуги без конечной вершины.

На рис. 2.24 представлены полугамаки и шлейфы графа G , показанного на рис. 2.23.



Рис. 2.23.

Вершина	Обязательные приемники	Фокус	Внутренние вершины	Входные вершины
v_3	v_2, v_7, v_5, t	v_2	v_3	v_3
v_2	v_7, v_5, t	v_7	v_3, v_2	v_3
v_7	v_5, t	v_5	v_7, v_6	v_7, v_6
v_6	v_5, t	v_5	v_7, v_6	v_7, v_6
v_5	t	t	v_7, v_6, v_5, v_4	v_7
v_4	t	t	v_7, v_6, v_5, v_4	v_7

Вершина	Полугамак	Шлейф
v_3		
v_2		
v_7		
v_5		

Рис. 2.24.

§ 4. Интервалы и сводимые графы

4.1. Подграф $I(v)$ называется *интервалом* с начальной вершиной v , если эта вершина принадлежит каждой зоне $I(v)$ и для всякой вершины w из $I(v)$, отличной от v , множество вершин Γ_w^{-1} содержится в $I(v)$. Из определения интервала вытекает, что любые два максимальных интервала графа не имеют общих вершин.

Выделение интервалов в графе происходит следующим образом. Пусть вершины графа как-то упорядочены, например с помощью M -нумерации. Берем первую вершину в качестве начальной вершины интервала (вначале это вход графа) и включаем в интервал

все те вершины, у которых все предшественники уже включены в интервал, а также все те контуры, которые проходят через данную вершину и у которых нет общих дуг с контурами, не проходящими через данную вершину. По завершении этого процесса каждая вершина будет включена в какой-нибудь, возможно тривиальный, интервал.

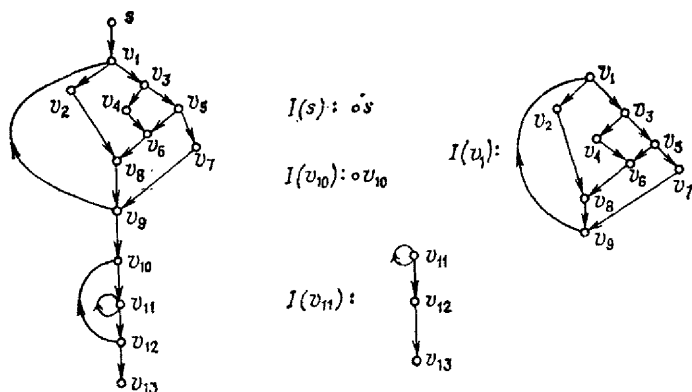


Рис. 2.25.

Пример. В графе на рис. 2.25 можно выделить четыре интервала: $I(s)$, $I(v_1)$, $I(v_{10})$ и $I(v_{11})$.

Граф G^* называется *производным* графом, если:

а) вершины графа G^* соответствуют максимальным интервалам графа G ;

б) вершина v графа G^* , соответствующая интервалу $I(v)$, соединяется дугой (v, w) с вершиной w , соответствующей интервалу $I(w)$, тогда и только тогда, когда в G имеется дуга из некоторой вершины интервала $I(v)$ в некоторую вершину интервала $I(w)$.

Другими словами, производный граф есть фактор-граф, но относительно интервалов, а не бикомпонент.

Последовательность $G_0 = G$, G_1, \dots, G_k , $k \geq 0$, где $G_1 = G^*$, а G_i есть производный граф для G_{i-1} , причем для любого i , $1 \leq i \leq k$, $G_{i+1} \neq G_i$, называется *интервальным представлением* графа G . Эта последовательность всегда конечна, т. е. для каждого графа существует предельный производный граф.

4.2. Граф G называется *сводимым*, если его предельный производный граф одновершинен.

Граф на рис. 2.25 является сводимым, так как его интервальное представление, показанное на рис. 2.26, имеет одновершинный предельный граф.

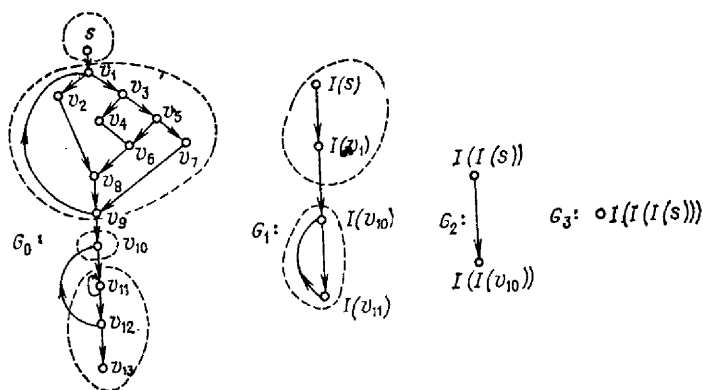


Рис. 2.26.

Существуют несводимые графы. Граф на рис. 2.27 несводим, так как его интервальное представление состоит всего из двух графов, последний из которых не является тривиальным.

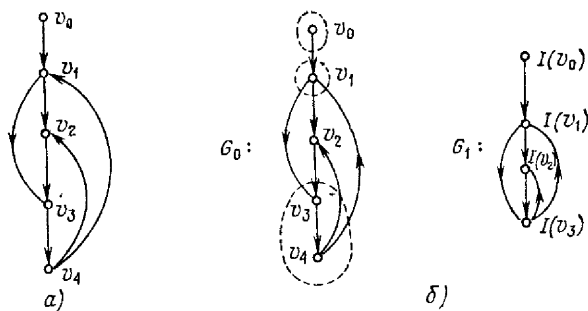


Рис. 2.27.

Лемма 20. *Сводимый граф аранжируем.*

Доказательство. Пусть некоторый граф G сводим, т. е. его предельный граф G_m , $m > 0$, представляет собой отдельную вершину. Рассмотрим процедуру упорядочения вершин графа G путем приписывания им различных последовательностей чисел. Порядок, устанавливаемый применением этой процедуры на вер-

пинах графа, зависит от порядка, в котором выбираются непосредственные преемники вершин в интервале. Тем не менее все эти результирующие упорядочения вершин сводимого графа являются аранжировками.

Единственной вершине графа G_m приписываем единицу, а для вершин каждого производного графа G_k , $k = m - 1, \dots, 0$, образуем последовательности так, что последовательность \mathcal{L}_p^k , приписываемая некоторой вершине p графа G_k , получается путем присоединения числа n_{m-k} (так называемого интервального номера вершины p , т. е. номера p в содержащем ее максимальном интервале $I(p)$) к последовательности $\mathcal{L}_{I(p)}^{k+1}$, приписанной вершине $I(p)$ графа G_{k+1} и уже образованной на предыдущем шаге.

Процесс нахождения интервальных номеров состоит в следующем. Головная вершина интервала имеет своим интервальным номером единицу. Следующее по порядку натуральное число получает в качестве интервального номера произвольная, незаномерованная вершина интервала, все предшественники которой уже имеют интервальные номера. (При организации этого процесса следует считать дуги, заходящие в начальную вершину интервала и принадлежащие зонам интервала, отсутствующими.) В силу определения интервала алгоритм заканчивает свою работу тогда и только тогда, когда все вершины интервала будут иметь различные интервальные номера. Покажем, что полученное упорядочение вершин есть аранжировка.

Предположим противное, т. е. предположим, что найдется пара вершин p и q в G , таких, что некоторый простой путь из входа s графа G в q содержит p и $\mathcal{L}_p^0 > \mathcal{L}_q^0$. Другими словами, для последовательностей $\mathcal{L}_p^0 = [n_0, n_1, \dots, n_m]$ и $\mathcal{L}_q^0 = [\bar{n}_0, \bar{n}_1, \dots, \bar{n}_m]$, приписанных соответственно вершинам p и q , существует такое k , $0 \leq k < m$, что $n_i = \bar{n}_i$ для $1 \leq i \leq k$ и $n_{k+1} > \bar{n}_{k+1}$. Из того, что первые k членов приписанных последовательностей совпадают, а $(k+1)$ -е различны, следует, что p и q входят в одну вершину I_{pq} графа G_{m-k} , но в разные вершины I_p и I_q графа $G_{m-(k+1)}$. Отсюда и из существования простого пути из входа графа G до q , содержащего p , а также учитывая то, что начальная вершина интервала является его единственной входной вершиной, непосредственно получаем следующее. Су-

существует путь в I_{pq} из I_p в I_q , и либо I_p — начальная вершина интервала I_{pq} , либо существует путь, содержащий начальную вершину интервала. Отсюда следует, что интервальный номер вершины I_p заведомо меньше интервального номера I_q , т. е. $n_{k+1} < \bar{n}_{k+1}$, что противоречит предположению. Таким образом, естественное упорядочение вершин сводимого графа является аранжировкой и сводимый граф аранжируем.

Более того, имеет место теорема (доказательство которой мы опускаем):

Теорема 12. *Свойства сводимости, аранжированности и одноходовости зон графа эквивалентны.*

4.3. Введем в рассмотрение два преобразования графов.

T1: удалить петлю в вершине v .

T2: заменить вершины v и w одной вершиной, если w не является входной вершиной и v — единственный предшественник вершины w , т. е. $\Gamma_w^{-1} = \{v\}$. Дуга (v, w) при этом преобразовании удаляется. Дуги, ранее исходившие из этих вершин, а также дуги, заходящие в w , заменяются дугами, исходящими из новой вершины и заходящими в нее.

Граф называется *сводимым по Хехту и Ульману*, если он может быть преобразован в пустой одновершинный граф с помощью последовательного применения преобразований T1 и T2.

Лемма 21. *Определения сводимого графа и сводимого по Хехту и Ульману эквивалентны.*

Доказательство. То, что интервально сводимый граф сводим в смысле Хехта и Ульмана, вытекает из возможности стягивания в одну вершину интервала, поскольку все контуры интервала могут быть стянуты в петли в начальной вершине интервала.

Если же граф сводим по Хехту и Ульману, то, взяв вершину, в которую граф оказался стянутым, и применяя преобразования, обратные преобразованиям T1 и T2, мы можем получить лишь граф, являющийся интервалом или в более общем случае графом, у которого каждая зона будет одноходовой.

Лемма 22. *Граф несводим тогда и только тогда, когда он содержит конструкцию, показанную на рис. 2.28, где волнистые линии соответствуют путям в графе.*

Нетрудно увидеть, что граф на рис. 2.28 содержит зону (в нашем случае контур) с двумя входами.

Назовем носителем $D(G)$ орграфа G максимальный бесконтурный суграф G . Если $T(G)$ — дерево поиска в глубину для орграфа G , то носитель $D(G)$ есть $T(G)$, дополненное прямыми и поперечными дугами. Так как $T(G)$ определяется неоднозначно, то и носитель в общем случае определяется неоднозначно.

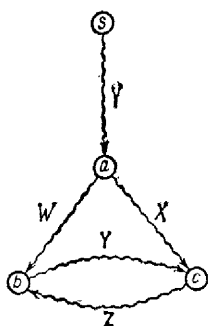


Рис. 2.28.

Теорема 13. *Граф сводим по Хехту и Ульману тогда и только тогда, когда он имеет единственный носитель.*

Доказательство: (\Rightarrow) Предположим, что граф сводим, но его носитель не единствен. Тогда G имеет по крайней мере два носителя $D_1(G)$ и $D_2(G)$. Так как D_1 и D_2 различны, то различны и их множества дуг, $E_1(D_1) \neq E_2(D_2)$. Не имеет места ни одно из включений $E_1 \subset E_2$ и $E_2 \subset E_1$ в силу максимальной носителя. Таким образом, $E_2 - E_1$ не пусто. Пусть (v, w) — дуга из $E_2 - E_1$. Она не может быть прямой дугой, так как все прямые дуги входят в E_1 , и не может быть поперечной по той же причине; следовательно, (v, w) — обратная дуга. Если $v = w$, то (v, w) — петля, что невозможно. Значит, $v \neq w$ и $v \neq s$, где s — вход G . Пусть $w = s$. Так как в D_2 существует путь μ из s в v , то μ вместе с дугой (v, w) образует контур, что невозможно. Значит, $w \neq s$. Кроме того, путь μ не содержит вершины w , иначе также был бы контур. Следовательно, дуга (v, w) не может принадлежать $E_2 - E_1$. Значит, $E_2 - E_1$ пусто, а отсюда следует единственность носителя для сводимого графа.

(\Leftarrow) Пусть G — несводимый граф. Докажем, что его носитель не единствен. Из леммы 22 следует, что G содержит по крайней мере одну конструкцию вида, показанного на рис. 2.28. Обозначим эту конструкцию через \mathcal{G} , а через V, W, X, Y, Z — соответственно множества дуг путей из s в a , из a в b , из a в c , из b в s и из c в b в какой-нибудь конструкции \mathcal{G} . Эти пять множеств не пересекаются, а V может быть пусто.

Построим теперь два различных носителя D_1 и D_2 для G . Пусть $T_1(G)$ есть дерево поиска в глубину для G , содержащее дуги $(V \cup W \cup Y \cup Z) - \{(d, b)\}$, где (d, b) — дуга в Z , заходящая в b . На рис. 2.29, а показано начало построения этого дерева. Z' на рисунке есть $Z \setminus \{(d, b)\}$. Аналогично, пусть $T_2(G)$ есть дерево

поиска в глубину, содержащее дуги $(V \cup X \cup Z \cup Y) - \{(e, c)\}$, где (e, c) есть дуга из Y , заходящая в c . На рис. 2.29, б показано начало построения этого дерева, Y' на рисунке есть $Y \setminus \{(e, c)\}$. Пусть D_1 есть $T_1(G)$ плюс его прямые и поперечные дуги; аналогично определим D_2 . Ясно, что D_1 и D_2 — носители графа G . Ни один из них не содержит всех дуг из $Y \cup Z$, так как иначе получился бы контур. Но D_1 и D_2 содержат разные подмножества $Y \cup Z$, и, таким образом, $D_1 \neq D_2$.

Будем говорить, что вершина v доминирует над вершиной w в G , если $v \neq w$ и каждый путь из входа s графа G в w содержит v .

Теорема 14. *Орграф G сводим тогда и только тогда, когда множество его дуг может быть разбито на два множества E_1 и E_2 таких, что E_1 порождает бесконтурный граф — носитель D графа G , а каждая дуга (v, w) из E_2 обладает свойством: либо $v = w$, либо w доминирует над v в G .*

Доказательство. (\Rightarrow) Предположим противное, т. е. что w не доминирует над v в G . Тогда в графе G имеется конструкция \mathcal{Z} , показанная на рис. 2.28, что противоречит предположению о сводимости G .

(\Leftarrow) Если для каждой обратной дуги (v, w) вершина w доминирует над v , то каждый контур в графе имеет один вход, а следовательно, граф сводим.

4.4. Проверка графа на сводимость. Пусть v и $w \neq s$ — две вершины в орграфе G с входом s . Втягиванием вершины w в v назовем преобразование графа, при котором удаляется вершина w и инцидентные ей дуги, добавляется дуга (v, x) вместо каждой удаленной (w, x) с $x \neq v$, если (v, x) не была ранее в графе, и добавляется (x, v) вместо каждой дуги (x, w) для $x \neq w$, если (x, v) не была ранее в графе.

Рассмотрим преобразование $T2'$: если (v, w) — единственная дуга, заходящая в w , и $w \neq s$, то втянем w в v .

Мы утверждаем, что граф сводим тогда и только тогда, когда он может быть преобразован в пустой одновершинный граф многократным повторением преобразования $T2'$. Это определение отличается от опреде-

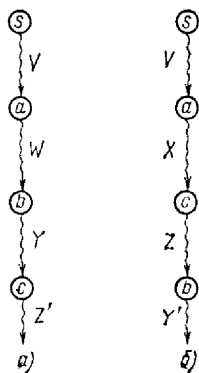


Рис. 2.29.

ления Хехта и Ульмана тем, что там допускаются петли и преобразования, приводящие к петлям. Легко увидеть, что граф с петлями сводим в смысле Хехта и Ульмана тогда и только тогда, когда он после удаления петель сводим в указанном выше смысле. Пусть нам известны полустепени захода каждой вершины. Выберем вершину, в которую заходит только одна дуга, и применим преобразование $T2'$, стягивая граф и изменяя число дуг, заходящих в каждую вершину. Каждое применение $T2'$ требует $O(n)$ времени и уменьшает каждый раз число вершин на единицу, так что алгоритм требует $O(n^2)$ времени. За счет использования более тонкого метода модификации информации после преобразования $T2'$ трудоемкость можно уменьшить до $O(m \log m)$.

Определим $C(w)$ как множество начал обратных дуг, заходящих в w , $P(w)$ — как множество вершин, из которых существует путь хотя бы в одну вершину из $C(w)$, не содержащий w . Если обратных дуг нет, то $C(w)$ и $P(w)$ пусты.

Лемма 23. *Граф G сводим тогда и только тогда, когда для всех w и всех $v \in P(w)$ в дереве поиска в глубину $T(G)$ существует путь из w в v .*

Доказательство. Для любых вершин v и w в том случае, когда v не является потомком w , в $T(G)$ существует путь из s в v , который не содержит w . Если существуют w и $v \in P(w)$ такие, что v не является потомком w , то существует путь из s в некоторую вершину в $C(w)$, не проходящий через w , и граф G несводим по теореме 14. Обратно, если G несводим, то по теореме 14 существуют обратная дуга (v, w) и путь из s в v , минуящий w . Тогда $s \in P(w)$ и s не является потомком w . Лемма доказана.

Пусть w есть вершина с наибольшим M -номером в G , в которую заходит обратная дуга. Предположим, что для всех $v \in P(w)$ существует путь из v в w по древесным дугам дерева $T(G)$. Пусть G' образован из G стягиванием всех вершин из $P(w)$ в w .

Лемма 24. *Каждая дуга (v', w') в G' соответствует дуге (v, w') в G , такой, что в $T(G)$ существует путь из v' в v .*

Доказательство. Пусть (v, w') — дуга в G . Если $w' \in P(w)$, то $v' \in P(w) \cup \{w\}$. Если $v \in P(w)$, то существует путь из v в w . Таким образом, либо (v, w') не соответствует ни одной дуге графа G' , либо (v, w')

соответствует дуге (v', w') графа G' с $v' = v$ или $v' = w$ и существует путь из w в v .

Пусть T' — подграф графа G' , дуги которого соответствуют дугам из $T(G)$.

Лемма 25. *T' с тем же упорядочением вершин, что и в $T(G)$, есть дерево поиска в глубину в G' . Обратные дуги соответствуют обратным дугам G , прямые дуги G' соответствуют прямым или поперечным дугам G , поперечные дуги G' — поперечным дугам G .*

Доказательство. Подграф дерева T , порожденный вершинами из $P(w) \cup \{w\}$, есть дерево. Каждая вершина в G' , исключая s , очевидно, имеет в точности одну дугу из T' , входящую в нее, так что T' есть дерево. Более того, упорядочение вершин в T' в точности то же, что и в T . Из леммы 24 следует, что обратная дуга из G соответствует либо обратной дуге из G' , либо ничему не соответствует; прямые дуги G либо ничему не соответствуют, либо соответствуют прямым дугам G' , а поперечные дуги G либо ничему не соответствуют, либо соответствуют поперечным или прямым дугам G' . Отсюда следует, что T' есть дерево поиска в глубину в G' .

Пусть множества $P'(x)$ и $C'(x)$ определены в графе G' так же, как множества $P(x)$ и $C(x)$ в графе G .

Лемма 26. *Для любой вершины $x < w$ в T' существует путь из x в y для всех $y \in P'(x)$ тогда и только тогда, когда в T существует путь из x в y для всех $y \in P(x)$.*

Доказательство. Предположим, что существует в $P'(x)$ вершина y , не являющаяся потомком x в T' . Тогда y не есть потомок x в T . Более того, существует путь p' в G' , не проходящий через x и идущий из y в некоторую вершину z' и такой, что в G' дуга (z', x) есть обратная дуга. По лемме 25 дуга (z', x) соответствует некоторой обратной дуге (z, x) в G . Подграф G' , порожденный вершинами из множества $P(w) \cup \{w\}$, сильно связен. Таким образом, существует не проходящий через x путь из y в z в графе G , состоящий из дуг, соответствующих дугам пути p' и, возможно, некоторым дугам между вершинами в $P(w) \cup \{w\}$; таким образом, $y \in P(x)$. Обратно, предположим, что существует вершина y в $P(x)$, не являющаяся потомком x в T . Если $y \notin P(w)$, то $y \in P'(x)$ и y не есть потомок x в T . Если $y \in P(w)$, то w не есть потомок x в T' и $w \in P'(x)$.

Доказанные леммы 23—26 лежат в основе эффективного алгоритма для проверки сводимости графа. Пусть T есть дерево поиска в глубину для графа G , и пусть $w_1 > w_2 > \dots > w_n$ — вершины из G , в которые заходят обратные дуги. Вычислим $P(w_1)$. Если в $P(w_1)$ найдется некоторая вершина, не являющаяся потомком w_1 , то стоп; в противном случае мы стягиваем вершины из $P(w_1)$ в w_1 для образования графа G' и вычисляем $P'(w_2)$ в G' . Если $P'(w_2)$ содержит вершину, не являющуюся потомком w_2 , то стоп; в противном случае мы формируем G'' стягиванием $P'(w_2)$ в w_2 и вычисляем $P''(w_3)$ в G'' . Граф сводим, если данной процедурой можно избавиться от обратных дуг. Тонкий момент в этом алгоритме — формирование графов G' , G'' и т. д. стягиванием вершин. Для построения алгоритма используем какую-нибудь эффективную процедуру объединения непересекающихся множеств. Множество с именем v будет содержать вершину v и все вершины, втянутые в v в текущем графе. Вначале будут существовать n множеств, каждое с именем, совпадающим с именем содержащейся в нем вершины. Функция НАЙТИ (x) будет возвращать имя множества, содержащего x , т. е. имя вершины, которая соответствует x в текущем графе. Процедура ОБЪЕДИНЕНИЕ (A, B, C) будет вычислять объединение множеств A и B (уничтожая A и B), давая новому множеству имя C . В дополнение к проверке сводимости алгоритм вычисляет для каждой вершины первую из вершин w_1, w_2, \dots, w_n , в которую x стягивается. Эта вершина обозначается $H(x)$ и $H(x)$ полагается равным нулю, когда x никогда не стягивается. $H(x)$ используется для конструирования последовательности преобразований $T2'$ для стягивания сводимого графа.

Алгоритм проверки графа на сводимость:

Вход: Граф $G = (V, E)$ с входом s и M -нумерацией вершин.

Выход: Ответ на вопрос: «Сводим или нет граф G ?»

начало

1. вычислить $ND(v)$ для каждой вершины v ;

примечание $ND(v)$ равна полустепени исхода вершины v **конец примечания**

2. для v от 1 до n шаг 1 цикл

3. **начало**

4. построить списки всех обратных, прямых и поперечных дуг, заходящих в v ;

5. сконструировать множество с именем v , содержащее v в качестве единственного элемента;

6. $H(v) := 0$;

7. конец цикла;

8. для w от n до 1 шаг — 1 цикл

9. начало

10. $P := \emptyset$;

примечание P будет содержать вершины из $P(w)$ в текущем графе конец примечания

11. для каждой обратной дуги (v, w) , заходящей в w , цикл

12. начало

13. включить НАЙТИ(v) в P ;

14. $Q := P$;

примечание множество Q будет содержать те вершины из множества P , чьи входящие дуги не были проверены конец примечания

15. пока $Q \neq \emptyset$ цикл

16. начало

17. выбрать вершину $x \in Q$ и удалить ее из Q ;

18. для каждой древесной, прямой или поперечной дуги (y, x) , заходящей в x , цикл

19. начало

20. $y' := \text{НАЙТИ}(y)$;

примечание к этому шагу все обратные дуги уже втянуты конец примечания

21. если $w > y' \vee w + \text{ND}(w) \leq y'$ то переход к шагу 34;

22. если $y' \notin P \wedge y' \neq w$ то

23. добавить y' к множествам P и Q ;

24. если $H(y') = 0$ то $H(y') := w$;

25. конец цикла;

26. конец цикла;

27. для $x \in P$ цикл

28. начало

29. ОБЪЕДИНЕНИЕ(x, w, w);

примечание это стягивание может порождать параллельные дуги, которые не удаляются, так как наличие их не влияет на алгоритм конец примечания

30. конец цикла;

31. конец цикла;

32. примечание если алгоритм достигает этого шага, то граф G сводим конец примечания

33. **примечание** если алгоритм достиг этого шага, то граф G несводим **конец** примечания

34. **конец** цикла;
конец

Легко доказать индукцией по числу вершин, в которые заходят обратные дуги, что эта процедура правильно проверяет сводимость графа. Это следует из лемм 23—26. Шаги 1—6 имеют трудоемкость $O(n + m)$. Каждая обратная дуга в G проверяется в точности один раз на шагах 11—31. Один раз вершина становится элементом множества P , втягивается в некоторую другую вершину, и составляющие ее вершины никогда не перепроверяются. Таким образом, каждая прямая, дуга или поперечная дуга проверяется в точности один раз на шагах 18—25. Каждая вершина становится элементом множества P самое большее один раз во время всех повторений цикла 8—34. Отсюда следует, что цикл 8—34 требует $O(n + m)$ шагов плюс $O(n)$ шагов для процедур ОБЪЕДИНЕНИЕ и $O(m)$ для НАЙТИ. Полная трудоемкость алгоритма определяется трудоемкостью операций над множествами, что составляет при эффективном способе построения объединения множеств $O(\min \{n \log n + m, m \log^* m\})$ или, более точно, $O(m\alpha(m, n))$, где $\alpha(m, n)$ — функция Аккермана.

4.5. **Стягивание сводимого графа.** Алгоритм, описанный в п. 4.4, неконструктивен, но мы можем использовать величины $H(v)$ для построения последовательности преобразований T_2' , которая будет стягивать сводимый граф. Для этого мы будем использовать N -номера вершин. Предположим теперь, что мы применяем алгоритм проверки на сводимость и каждой вершине v сопоставляем пару $(H(v), N(v))$. К концу алгоритма мы упорядочим вершины так, что вершина, помеченная парой (x_1, y_1) , будет встречаться прежде, чем помеченная парой (x_2, y_2) , тогда и только тогда, когда $x_1 > x_2$ или $x_1 = x_2$ и $y_1 < y_2$. Это упорядочение вершин назовем *редукционным порядком*. Трудоемкость процедуры — $O(n)$.

Лемма 27. Если граф G сводим, то мы можем втянуть вершины G путем использования преобразования T_2' в соответствии с их редукционным порядком.

Докажем лемму индукцией по числу втянутых вершин. Предположим, что все вершины после вершины v в редукционном упорядочении могут быть втянуты.

Это порождает граф G' , являющийся редукцией G . Рассмотрим вершину v . Если это не вход графа, то в вершину v заходит единственная древесная дуга. Если в графе G имеется обратная дуга (v, w) , такая, что из v в w идет путь, то все вершины x в пути по дереву от v к w могут быть втянуты прежде v , так как $H(x) \geq M(w)$ и $H(v) < M(v) \leq M(w)$. Если граф G содержит прямую дугу (v, w) , такую, что из v в w идет путь, то $H(w) < M(v)$ по леммам 23 и 26. Более того, $H(x) \geq H(w)$ и $N(x) \leq N(w)$ для всех вершин x на пути по дереву из v в w . Отсюда следует, что все вершины на пути по дереву из v в w втягиваются прежде, чем втягивается w . Предположим, что G содержит поперечную дугу (v, w) такую, что из v в w идет путь. Пусть x есть общий предок вершин v и w с наибольшим M -номером. Тогда $H(w) \leq M(x)$ по леммам 23 и 26, и все вершины y на пути по дереву из x в w и из x в v удовлетворяют соотношениям $H(y) \geq H(v)$ и $N(y) \leq N(w)$. Таким образом, все вершины на этих путях втягиваются ранее w . Отсюда следует, что в G' вершина v может иметь только одну дугу, входящую в нее, и поэтому она может быть втянута. Лемма доказана.

На рис. 2.31 приведены значения переменных $H(v)$ и N -номера вершин для графа, показанного на рис. 2.30. Порядок редукции: $A, D, E, H, B, C, G, J, F, I, K$.

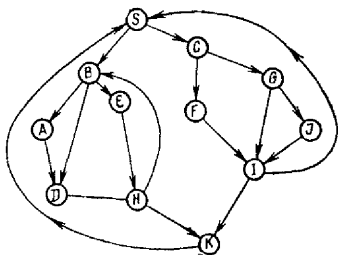


Рис. 2.30.

4.6. Строгой последовательностью вершин графа называется такая последовательность вершин (возможно, с повторениями), что любой простой путь в графе есть ее подпоследовательность. **Слабой последовательностью** вершин

называется последовательность, содержащая в качестве подпоследовательности любой базисный путь в графе, т. е. такой путь $\mu = [v_1, \dots, v_k]$, что никакая его собственная подпоследовательность не является путем из v_1 в v_k . Очевидно, что каждая строгая последовательность одновременно является слабой.

Теорема 15. *Каждый сводимый граф с k вершинами имеет строгую последовательность вершин длины не более чем $k + ck \log k$, где $c = 3/(\log^3 2) = 5,13$.*

Для доказательства теоремы проведем ряд вспомогательных построений.

Марковский и Тарьян показали, что для бесконечно многих значений n существуют сводимые графы с n вершинами и полустепенями захода и исхода, не превосходящими двух, для которых слабая последовательность вершин имеет длину по крайней мере $(1/2) \log n - O(n \log n)$.

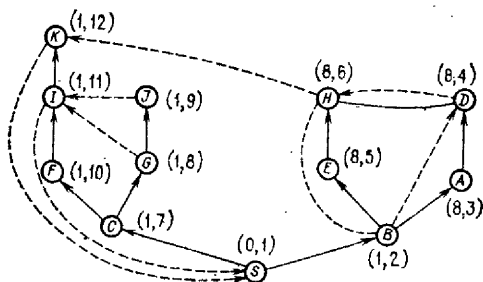


Рис. 2.31.

Областью $R(V', E')$ с головной вершиной h сводимого графа $G = (V, E)$ называется суграф, порождаемый множеством вершин V' и дуг $E' \subseteq V' \times V'$, $h \in V'$, удовлетворяющий свойствам:

а) если $(v, w) \in E$ и $w \in V'$, причем $w \neq h$, то $v \in V'$;

б) если $(v, w) \in E$, $v \in V'$ и $w \neq h$, то $(v, w) \in E'$.

Таким образом, единственный способ попасть в область — это пройти через головную вершину. Заметим также, что E' содержит все дуги из $V' \times V'$, за исключением тех, которые заходят в головную вершину, начинаясь внутри области.

Лемма 28. Пусть $R = (V', E')$ есть область с головной вершиной h сводимого графа $G = (V, E)$ с входом s . Пусть μ есть простой путь в G , который начинается в некоторой вершине вне R . Тогда μ не содержит ни одной обратной дуги области R .

Доказательство. Пусть μ содержит две различные вершины v и w , такие, что соединяющая их дуга (v, w) — обратная. По определению области путь μ достигает вершины h раньше, чем v или $h = v$. Так как G есть граф с входом s , то существует путь ν из s в h , и мы можем предположить без ограничения общности, что путь ν — простой. Таким образом, ни одна вершина области R , исключая h , не входит в ν снова.

Тогда путь ν , следуя по той части μ , которая ведет из h в ν , образует путь из s в ν , который минует w , если только w не совпадает с h . В первом случае мы получаем противоречие с теоремой 14, которая утверждает, что w доминирует над ν . Во втором случае w дважды встречается в пути μ , так как $h = w = \nu$ исключается нашим предположением. Лемма доказана.

Из теоремы 14 следует, что существует такое упорядочение вершин сводимого графа, что любой путь, не использующий обратных дуг, есть подпоследовательность упорядоченной последовательности вершин. Назовем такое упорядочение *ациклическим*. Из леммы 28 следует, что если путь входит в область R через головную вершину, то он должен следовать по подпоследовательности ациклического упорядочения, пока не покинет эту область.

Заметим, что когда мы стягиваем сводимый граф с помощью преобразований T_1 и T_2 , вершины постоянно соответствуют областям. Применение T_1 не увеличивает числа вершин в области, хотя и добавляет некоторые дуги. Таким образом, только T_2 строит области с постоянно растущим числом вершин.

Лемма 29. Пусть $R = (V', E')$ — область графа G , изображаемая некоторой вершиной при стягивании G , причем V' не есть одноэлементное множество. Тогда V' может быть разбито на два непустых непересекающихся подмножества вершин V_1 и V_2 , таких, что (V_1, E_1) и (V_2, E_2) — области, где $E_1 = E' \cap (V_1 \times V_1)$ и $E_2 = E' \cap (V_2 \times V_2)$.

Введем теперь специальный класс сводимых графов, так называемые *спиральные графы*, для которых легко строятся требуемые последовательности вершин. Так как любой сводимый граф достаточно просто преобразуется в спиральный, то последовательность вершин для спирального графа также просто преобразуется в последовательность вершин для исходного. Класс спиральных графов определяется рекурсивно.

1) Отдельная вершина есть спиральный граф.

2) Если $G = (V, E)$ есть спиральный граф с входом s и ν — новая вершина, то:

а) $(V \cup \{\nu\}, E \cup E' \cup \{(v, s)\})$ есть спиральный граф с входом s (здесь E' — множество дуг из вершин $w \in V$ в ν);

б) $(V \cup \{\nu\}, E \cup E' \cup \{(v, s)\})$ есть спиральный граф с входом ν .

3) Граф, отличный от определенного выше, не является спиральным.

На рис. 2.32 изображены две конструкции спиральных графов, соответствующих двум альтернативам п. 2) определения.

При построении спирального графа вершины добавляются к нему в определенном порядке. При этом все



Рис. 2.32.

дуги, исключая дуги (v, s) , добавляемые в случае 2), «ввинчиваются в него», т. е. вначале к графу присоединяются их концы, а потом уже начала, поэтому начала дуг располагаются дальше от центра.

Вершину v , добавляемую к спиральному графу по правилу 2а), будем называть *ведомой* вершиной, а добавляемую по правилу 2б) —

ведущей вершиной. Вход спирального графа — всегда ведущая вершина.

Лемма 30. 1. *Каждый спиральный граф сводим; все спиральные графы, из которых он образован, представляют собой его области.*

2. *Дуги, добавляемые по правилу 2а), исключая (v, s) , и дуга (v, s) , добавляемая по правилу 2б), — прямые, остальные — обратные.*

3. *Каждая ведущая вершина доминирует над всеми ранее добавленными.*

Доказательство. 1. По индуктивному предположению граф на рис. 2.32, а или 2.32, б сводим. В процессе стягивания его в одну вершину мы придем к паре вершин с дугами между ними. Этот граф, очевидно, сводим.

2. Дуги, обозначенные как прямые, образуют бесконтурный граф, и никакая дуга не может быть добавлена без образования контура. По теореме 14 и части 1 леммы разделение дуг на прямые и обратные единственно.

3. По части 2 леммы обратные дуги входят в ведущие вершины из всех ранее добавленных вершин. Поэтому утверждение 3 вытекает из теоремы 14.

Дадим теперь рекурсивный метод конструирования последовательности вершин для спиральных графов.

Лемма 31. Пусть $\mu = [v_1, \dots, v_k]$ — простой путь в сводимом графе, и пусть $(v_{i_1-1}, v_{i_1}), (v_{i_2-1}, v_{i_2}), \dots, (v_{i_r-1}, v_{i_r})$ — последовательность обратных дуг вдоль μ именно в таком порядке. Тогда v_i доминирует над v_{i_j-1} для всех $j, 1 \leq j \leq k$.

Доказательство леммы опускаем.

Так как отношение доминирования транзитивно, то из леммы 31 следует, что каждое начало обратной дуги вдоль простого пути доминирует над всеми предыдущими началами обратных дуг.

Лемма 32. Пусть G — спиральный граф, образованный вершинами v_1, v_2, \dots, v_k , добавлявшимися именно в таком порядке. Пусть G' — спиральный граф, состоящий из вершин v_1, v_2, \dots, v_{j-1} и дуг между ними в G . Пусть G'' — спиральный граф, образованный вершинами v_j, v_{j+1}, \dots, v_k и всеми дугами между ними. Пусть A и B — последовательности вершин для G' и G'' , и пусть \hat{A} и \hat{B} — множества вершин соответственно G' и G'' , взятых в ациклическом порядке. Тогда $A\hat{B}\hat{B}\hat{A}B$ есть последовательность вершин для графа G .

Доказательство. Предположим, что простой путь p начинается в G' и потом заходит в G'' . Если p проходит более чем одну обратную дугу до того или после того, как он покинет G' , он никогда не сможет вернуться в G' . Из доказательства леммы 30 (часть 2) вытекает, что каждая обратная дуга входит в ведущую вершину. По лемме 31 вторая обратная дуга входит в вершину, которая доминирует над началом h предыдущей обратной дуги. По лемме 30 (часть 3), примененной в вершине h , головная вершина в G' не может быть достигнута без повторного прохождения h . Таким образом, если путь p достигает G' и возвращается, то он проходит в точности через одну обратную дугу. Часть пути p до возвращения в G' есть, таким образом, подпоследовательность $A\hat{B}\hat{B}$. При прохождении через G' во второй раз путь p не может проходить через обратные дуги по лемме 28. Таким образом, путь p есть подпоследовательность $A\hat{B}\hat{B}\hat{A}$, пока он снова не покинет G' . В этом случае p не может вообще вернуться в G' , так как тогда он должен был бы пройти через две ведущие вершины, которые доминируют над головной вершиной G' , что невозможно. Таким образом, путь p есть подпоследовательность $A\hat{B}\hat{B}\hat{A}B$. Случай, когда

путь p начинается в G'' , а не в G' или когда p начинается в G' , но никогда не возвращается, легко сводятся к рассмотренному.

Нам нужно расширить лемму 32, чтобы применить ее для разбиения спирального графа на три части, одна из которых состоит из одной вершины.

Лемма 33. Пусть граф G определен, как и в лемме 32, подграф G' порожден вершинами v_1, \dots, v_{j-1} и G'' порожден вершинами v_{j+1}, \dots, v_k . Пусть A и B — последовательности вершин для графов G' и G'' , и пусть \hat{A} и \hat{B} — ациклические упорядочения этих графов. Тогда $A\hat{B}\hat{B}\hat{A}v_j\hat{B}\hat{B}\hat{A}B$ есть последовательность вершин для G .

Лемма 33 является непосредственным обобщением леммы 32. Теперь нужно показать, что при произвольном приписывании весов существует последовательность вершин, в которой вершины с наибольшим весом будут встречаться меньшее число раз. Причина, заставляющая рассматривать веса, состоит в том, что произвольный сводимый граф будет стягиваться в подграфы спирального. При таком подходе вершины спирального графа будут изображать области разнообразного размера и вес вершин будет соответствовать размеру изображаемой области.

В оставшейся части параграфа используем обозначения для констант: $a = 2/\log(3/2)$ и $b = (3/2)^{1/2}$. Заметим, что $a \log b = 1$.

Лемма 34. Пусть G — спиральный граф, порожденный вершинами v_1, \dots, v_k именно в таком порядке.

Пусть v_i имеет вес ω_i , и пусть $\Omega = \sum_{i=1}^k \omega_i$. Тогда G имеет последовательность вершин, в которой v_i встречается самое большее $a \log(b\Omega/\omega_i)$ раз, $i \in [1, k]$.

Для $k = 1$ утверждение очевидно. Предположим, что $k > 1$ и что лемма справедлива для спиральных графов с числом вершин, меньшем k . Легко показать, что может встретиться один из случаев:

1. Для некоторого j , $1 < j \leq k$, имеем

$$(1/3)\Omega \leq \sum_{i=1}^{j-1} \omega_i \leq (2/3)\Omega.$$

2. Для некоторого j

$$\omega_j > (1/3)\Omega, \quad \sum_{i=1}^{j-1} \omega_i < (1/3)\Omega, \quad \sum_{i=j+1}^k \omega_i < (1/3)\Omega.$$

1. $(1/3)\Omega \leq \sum_{i=1}^{j-1} \omega_i \leq (2/3)\Omega$. Разбиваем граф G по лемме 32, полагая, что G' состоит из вершин v_1, \dots, v_{j-1} , а G'' — из оставшихся вершин. Пусть $\Omega' = \sum_{i=1}^{j-1} \omega_i$ и $\Omega'' = \sum_{i=j}^k \omega_i$. По индуктивному предположению, примененному к G' и G'' , существует последовательность вершин A для G' , в которой v_i встречается самое большее $a \log(b\Omega'/\omega_i)$ раз для $1 \leq i < j$. Точно так же существует последовательность вершин для G'' , в которой v_i встречается самое большее $a \log(b\Omega''/\omega_i)$ раз для $i \in [j:k]$. По лемме 32 существует последовательность вершин для G , в которой для $i \in [j:k]$ вершина v_i встречается самое большее $2 + a \log(b\Omega''/\omega_i) = a \log(2^{2/a}b\Omega''/\omega_i)$ раз. Так как $\Omega'' \leq (2/3)\Omega$ и $(2/3)2^{2/a} = 1$, вершина v_i встречается самое большее $a \log(b\Omega/\omega_i)$ раз. Аналогичные рассуждения применимы и в случае $i \in [1:j]$.

2. $\omega_j > (1/3)\Omega$, $\sum_{i=1}^{j-1} \omega_i < (1/3)\Omega$, $\sum_{i=j+1}^k \omega_i < (1/3)\Omega$. Разобьем граф G на G' , v_j и G'' , как в лемме 33, положив, что G' состоит из v_1, v_2, \dots, v_{j-1} и G'' — из вершин $v_{j+1}, v_{j+2}, \dots, v_k$. По индуктивному предположению и из того факта, что $\sum_{i=1}^{j-1} \omega_i$ и $\sum_{i=j+1}^k \omega_i$ меньше, чем $\Omega/3$, найдутся последовательности вершин A и B для G' и G'' , такие, что v_i встречается самое большее $a \log(b\Omega/3\omega_i)$ раз для $i \neq j$. По лемме 33 найдется последовательность вершин для G , в которой v_i встречается самое большее

$$4 + a \log(b\Omega/3\omega_i) = a \log(2^{4/a}b\Omega/3\omega_i)$$

раз. Так как $(1/3)2^{4/a} = 3/4$, то мы получаем в этом случае требуемый результат. Наконец, существует одно вхождение вершины v_j в последовательность вершин для G . Так как $a \log(b\Omega/\omega_j) \geq a \log b = 1$, доказательство закончено.

4.7. Построение последовательности вершин. В основе построения последовательности вершин длины $O(n \log n)$ для любого сводимого графа лежит разбиение графа на куски, причем никакой из них не превосходит двух третей целого графа. Куски сами по себе — это области, и последовательности вершин для

них находятся рекурсивно. Потом мы образуем подграф спирального графа стягиванием каждой из этих областей R в отдельную вершину v_R . Желаемая последовательность вершин находится, если взять последовательность вершин для спирального графа и подставить ациклически упорядоченные последовательности вершин области R на место вершины v_R .

Лемма 35. Пусть $G = (V, E)$ — сводимый граф с входом s и с $k > 1$ вершинами. Тогда мы можем найти множество непересекающихся областей R_1, \dots, R_m , объединение которых включает все вершины графа G , причем:

1) ни одна из областей R_1, \dots, R_m не имеет больше чем $(2/3)k$ вершин;

2) существует последовательность областей S_1, \dots, S_m , такая, что:

а) $S_1 = R_1$;

б) для $i > 1$ S_i состоит из областей S_{i-1} и R_i , причем одна предшествует другой;

в) S_m есть G ;

3) граф, образованный из G стягиванием каждой из областей R_1, \dots, R_m в отдельную вершину без петель, есть спиральный граф, в котором, быть может, не удалена ни одна дуга.

Доказательство. По лемме 29 каждая область более чем с одной вершиной есть объединение двух областей, одна из которых есть предшественник другой. Заметим, что если T — область более чем с $(2/3)k$ вершинами, то:

1) или она составлена из двух непустых областей, одна из которых имеет более чем $(2/3)k$ вершин,

2) или она составлена из двух областей, большая из которых имеет от $(1/3)k$ до $(2/3)k$ вершин.

Рассмотрим алгоритм:

начало

$T := G$;

пока T имеет больше чем $(2/3)k$ вершин цикл

начало

пусть T составлена из областей T_1 и T_2 , причем T_1 имеет не меньше вершин, чем T_2 ;

печать (T_1, T_2) ;

$T := T_1$;

конец

печать (T, T) ;

конец

Этот алгоритм будет производить последовательность пар $(S_m, R_m), (S_{m-1}, R_{m-1}), \dots, (S_1, R_1)$, что доказывает части 1) и 2) леммы.

Часть 3) докажем индукцией по i . А именно докажем, что после редукции S_i есть спиральный граф с отсутствующими, быть может, некоторыми дугами. Базис индукции $i=1$ тривиален. Если R_i есть предшественник S_{i-1} при образовании S_i , то результат непосредственно следует из конструкции 2б) определения спирального графа. Если же S_{i-1} есть предшественник для R_i , то выполняется конструкция 2а).

Пример. Рассмотрим граф на рис. 2.33. Он составлен из областей $\{1, 2\}$ и $\{3, 4, \dots, 10\}$. Последняя имеет более чем $(2/3) \times 10$ вершин, так что мы печатаем пару $(S_m, R_m) = (\{1, 2, \dots, 10\}, \{1, 2\})$. Потом мы работаем с областью $\{3, 4, \dots, 10\}$, которая может быть разбита одним из двух способов: либо выделением вершины 3, либо вершины 10. Предположив последнее, получаем пару $(S_{m-1}, R_{m-1}) = (\{3, \dots, 10\}, \{10\})$. Потом мы разделяем область $\{3, 4, \dots, 9\}$ на $\{3\}$ и $\{4, 5, \dots, 9\}$. Первая область есть R_{m-2} , а последняя имеет не более чем $(2/3) \times 10$ вершин, так же как и S_{m-1} и R_{m-1} . Таким образом, получаем:

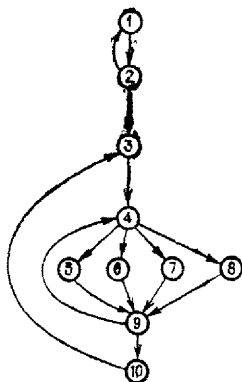


Рис. 2.33.

i	R_i	S_i
1	$\{4, 5, \dots, 9\}$	$\{4, 5, \dots, 9\}$
2	$\{3\}$	$\{3, 4, \dots, 9\}$
3	$\{10\}$	$\{3, 4, \dots, 10\}$
4	$\{1, 2\}$	$\{1, 2, \dots, 10\}$

Лемма 36. Пусть G — сводимый граф, разбитый на области R_1, \dots, R_m , как в лемме 35. Пусть R_i имеет последовательность вершин A_i и циклическое упорядочение \hat{A}_i для $i \in [1 : m]$. Пусть H — спиральный граф, построенный из G стягиванием каждой области R в отдельную вершину и добавлением некоторых дуг (для получения спирального графа). Пусть B — последовательность вершин для H . Пусть последовательность C построена из B заменой в B каждого вхождения

вершины v_i из H , изображающей R_i , на ациклическую упорядоченность \hat{A}_i . Тогда $A_1 A_2 \dots A_m C$ есть последовательность вершин для графа G .

Доказательство. Пусть p — простой путь в G . Мы можем записать p как $p_1 p_2$, где p_1 состоит из префикса p , т. е. его части от начала до того места, когда p покидает одну из областей R_1, \dots, R_m , в которой он начинается. Конечно, p_1 есть подпоследовательность последовательности $A_1 A_2 \dots A_m$. Рассмотрим путь q в H , состоящий из тех вершин в графе H , которые представляют области R_1, \dots, R_m и через которые проходит p_2 в G . Путь q должен быть простым, иначе p_2 будет заходить в одну и ту же область дважды. Так как области могут посещаться только через их головные вершины, то будет нарушена ациклическость пути p_2 и, следовательно, пути p . Таким образом, q есть подпоследовательность B и по лемме 28 p_2 есть подпоследовательность C . Таким образом, p есть подпоследовательность последовательности $A_1 A_2 \dots A_m C$.

Доказательство теоремы 15 проведем индукцией по числу вершин k . Базис индукции $k = 1$ очевиден. Предположим, что утверждение теоремы справедливо для графов с числом вершин меньше k . По лемме 36 оценим сверху длину последовательности вершин $A_1 A_2 \dots A_m C$. Пусть ω_i — число вершин в области R_i . Тогда по индуктивному предположению A_i имеет длину, не превышающую $\omega_i + c \omega_i \log \omega_i$. По лемме 34 спиральный граф H , образованный из G так, как указано в лемме 36, имеет последовательность вершин B , в которой вершина, изображающая R_i , встречается не более, чем $a \log (bk/\omega_i)$ раз. Таким образом, последовательность вершин C имеет длину самое большее $\sum_{i=1}^m a \omega_i \log (bk/\omega_i)$. Следовательно, $A_1 A_2 \dots A_m C$ имеет длину, ограниченную сверху выражением

$$\begin{aligned} \sum_{i=1}^m (\omega_i + c \omega_i \log \omega_i + a \omega_i \log (bk/\omega_i)) &= \\ &= \sum_{i=1}^m \omega_i + \sum_{i=1}^m (c - a) \omega_i \log \omega_i + a \sum_{i=1}^m \omega_i \log bk = \\ &= k + \sum_{i=1}^m (c - a) \omega_i \log \omega_i + ak \log k + ak \log b. \quad (*) \end{aligned}$$

Так как $\omega_i \leq (2/3)k$ для всех i , то по лемме 35 имеем

$$\sum_{i=1}^m (c - a) \omega_i \log \omega_i \leq \sum_{i=1}^m (c - a) \omega_i \log (2/3) k = \\ = (c - a)(k \log k - k \log (3/2)). \quad (**)$$

Подставляя $(**)$ в $(*)$, получаем величину $k + ck \log k - ck \log (3/2) + ak \log (3/2) + ak \log b$. Теперь достаточно заметить, что $-c \log (3/2) + a \log (3/2) + a \log b = -3 + 2 + 1 = 0$ при нашем выборе a, b и c .

Теорема 16. *Последовательность вершин для сводимого графа с n вершинами и не более чем с $2n$ дугами может быть построена с трудоемкостью $O(n \log n)$.*

Доказательство. Достаточно просто проверить, что конструкции лемм 32—34, 36 и теоремы 15 требуют времени, пропорционального длине порождаемой последовательности. Остается задача, рассматриваемая в лемме 35, где было доказано существование последовательности областей R_1, \dots, R_m без указания на способ ее эффективного построения. Но так как описанный в п. 4.4 алгоритм проверки графа на сводимость требует $O(n \log n)$ времени, то построение нужной последовательности R_1, \dots, R_m может быть также проведено за то же время.

§ 5. Контур в орграфах

Одной из важнейших задач, связанных с контурами, является задача нахождения множества всех контуров. Трудность ее состоит прежде всего в том, что число контуров орграфа может быть экспоненциально большим относительно числа вершин. Поэтому при разработке алгоритмов внимание обращается не на полную трудоемкость алгоритма, а на относительную, т. е. на трудоемкость, приходящуюся на один контур.

5.1. Алгоритм отыскания контуров с использованием матрицы достижимости. Пусть заданы матрица смежности $A(G) = \|a_{ij}\|$ и матрица достижимости $R(G) = \|r_{ij}\|$ орграфа G . Алгоритм начинается с того, что рисуется вершина и ей присваивается метка «1». Затем для каждого i , такого, что $a_{1i}r_{i1} = 1$, $i \in [1:n]$, рисуется дуга и концевые вершины полученных дуг помечаются метками « i ». Далее берем некоторую висячую вершину полученного корневого ордерова

с меткой « k » ($k \neq 1$), рисуем дугу из этой вершины для каждого j , такого, что $a_{kj}r_{j1} = 1$, $j \in [1:n]$, и присваиваем метку « j » концевой вершине такой дуги. Заметим, однако, что дугу нельзя построить, если $j \neq 1$ и при этом j уже выступала в роли некоторой вершины на пути из корня дерева к вершине, помеченной меткой « k ». Продолжаем этот процесс до тех пор, пока все концевые вершины не будут иметь метку «1». Итак, получены все пути от корня дерева до концевой вершины для каждого простого контура, проходящего в орграфе G через вершину v_1 . Эти контуры определяются метками при вершинах в указанных путях. Теперь образуем подматрицы A_2 и R_2 , удаляя из A и R первую строку и первый столбец. Аналогичная процедура, примененная к A_2 и R_2 , найдет контуры, проходящие через вершину v_2 , но не проходящие через вершину v_1 . На третьем шаге исключим из A и R по две первых строки и по два первых столбца. Работая с полученными подматрицами A_3 и R_3 , найдем контуры, проходящие через вершину v_3 , и т. д. Алгоритм применим в случаях, когда матрица достижимости уже известна или когда граф задается с помощью K -формулы.

Пример приведен на рис. 2.34.

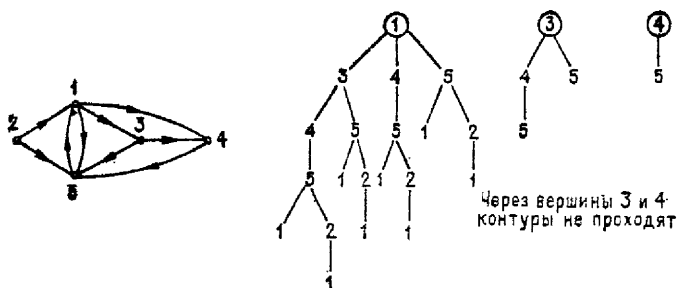


Рис. 2.34

5.2. Алгоритм Джонсона отыскания контуров. Пусть орграф задан списками смежности $A_G(v)$, причем обращение к вершинам происходит по их M -номерам. Работа алгоритма основана на поиске в глубину. Поиск начинается в некоторой вершине s_0 , в процессе его строится путь $[s_0, v_1, v_2, \dots, v_k]$ такой, что $M(v_i) > M(s_0)$ или, в силу нашей договоренности, $v_i > s_0$, $i \in [1:k]$. Контур объявляется построенным, если сле-

дующая вершина v_{k+1} есть s_0 . В процессе поиска вершины рассматриваемого пути хранятся в стеке — текущем списке. Вершина v помещается в стек при вызове процедуры $\text{КОНТУР}(v)$ и удаляется после возвращения из процедуры. Чтобы предупредить возможность прохождения контуров, начинающихся в какой-либо вершине пути, отличной от s_0 , необходимо указать, что все вершины из текущего списка (кроме s_0) не могут быть использованы дважды для его удлинения. С этой целью вершина при помещении в стек блокируется путем присвоения специальной переменной $b(v)$ значения *истина*. После удаления вершины из стека она не обязательно разблокируется, особенно когда возвращение из процедуры не связано с построением контура. Задержка с разблокированием в этом случае позволяет избежать поиска контура в той части графа, в которой такой поиск раньше был безуспешен. Для недоступных (блокированных) вершин формируются списки предшественников $B(u)$. Вершины, находящиеся в них, также недоступны и становятся доступными одновременно с разблокированием исходной вершины.

процедура $\text{КОНТУР}(\text{целый } v, \text{ логический } f);$

1. **начало**
2. $f := \text{ложь};$
3. поместить вершину v в стек;
4. $b(v) := \text{истина};$
5. **для** $w \in A(v)$ **цикл**
6. **начало**
7. **если** $w = s$ **то**
8. **начало**
9. вывести контур, состоящий из вершин в стеке, начиная с вершины s , без изменения самого стека;
10. $f := \text{истина};$
11. **конец**
12. **иначе если** $b(w)$ **то**
13. $\text{КОНТУР}(w, g);$
14. $f := f \vee g;$
15. **конец цикла;**
16. **если** f **то** $\text{РАЗБЛОК}(w)$
17. **иначе для** $w \in A(v)$ **цикл** $B(w) := B(w) \cup \{v\};$
18. удалить v из стека;
19. **конец процедуры**

При удлинении пути $[s, v_1, v_2, \dots, v_k]$ дугой (v_k, v_{k+1}) возможны три случая:

1. Если $v_{k+1} = s$, то это означает, что построен новый контур $[s, v_1, v_2, \dots, v_k, s]$. Выводим этот контур и присваиваем переменной f значение *истина*, указывая тем самым, что найден контур с начальной вершиной s , проходящий через v_k .

2. Если $v_{k+1} \neq s$ и v_{k+1} не заблокирована, то добавляем v_{k+1} к рассматриваемому пути посредством рекурсивного обращения к процедуре КОНТУР.

3. Если вершина v_{k+1} заблокирована, то пропускаем ее и ищем другую дугу, начинающуюся в вершине v_k . Если неисследованных дуг, начинающихся в вершине v_k , нет, то v_k удаляется из стека, и мы возвращаемся назад в вершину v_{k-1} . При этом мы делаем вершину v_k снова доступной, если она привела к построению контура (f имеет значение *истина*). В противном случае она остается недоступной и добавляется в каждое из множеств $B(w)$ для всех вершин w — потомков v_k .

Для того чтобы вершину v разблокировать, нужно изменить значение переменной $b(v)$ на *ложь*. Однако, если вершина v становится доступной, то же самое происходит со всеми ее недоступными предшественниками, не принадлежащими рассматриваемому пути. Таким образом, все элементы множества $B(v)$ должны быть сделаны доступными, и $B(v)$ становится пустым.

Процесс разблокирования вершин производится процедурой РАЗБЛОК:

процедура РАЗБЛОК(u);

1. начало

2. $b(u) := \text{ложь}$;

3. для $w \in B(u)$ цикл

4. начало

5. $B(u) := B(u) \setminus \{w\}$;

6. если $b(w)$ то РАЗБЛОК(w);

7. конец цикла;

8. конец

Поскольку любой контур целиком лежит внутри некоторой бикомпоненты, то будем считать, что в графе уже выделены бикомпоненты. Это позволяет применять процедуру КОНТУР к одной из бикомпонент. В итоге алгоритм приобретает вид:

Вход: Граф $G = (V, E)$ с выделенными бикомпонентами, заданный списками смежности $A(v)$.

Выход: Список контуров.

начало

1. $s := 1$;

примечание вначале стек пустой **конец** **примечания**

2. **пока** $s < n$ **цикл**

3. **начало**

4. пусть $S = (V_s, E_s)$ — нетривиальная бикомпонента, содержащая вершину с наименьшим номером в подграфе, порожденном вершинами $\{s, s+1, \dots, n\}$, и пусть $A_s(v)$ есть список смежности вершины v в S ;

5. **если** $A_s \neq \emptyset$ **то**

6. $s :=$ вершина с наименьшим номером в V_s ;

7. **для** $v \in V_s$ **цикл**

8. **начало**

9. $b(v) :=$ ложь;

10. $B(v) := \emptyset$;

11. $\text{КОНТУР}(s, f)$;

12. $s := s + 1$;

13. **конец** **цикла**;

14. **иначе** $s := n$;

15. **конец** **цикла**;

конец

После обращения к процедуре $\text{КОНТУР}(s, f)$ будут порождены все контуры, содержащие вершину s , и ее можно будет удалить из графа. Это, конечно, сильно изменит бикомпоненты графа. Корректность алгоритма вытекает из леммы, которую мы приводим без доказательства.

Лемма 37. Для любой вершины $x \neq s$ существует обращение к процедуре $\text{РАЗБЛОК}(v)$, которая присваивает переменной $b(x)$ значение ложь тогда и только тогда, когда:

а) *из x в s существует путь, содержащий v , для которого только v и s принадлежат стеку;*

б) *из x в s не существует пути, для которого только v и s принадлежат стеку.*

Алгоритм Джонсона имеет трудоемкость $O((n+m) \times (c+1))$ для графа с n вершинами, m дугами и c простыми контурами, т. е. практически линейную относительно числа дуг графа на контур.

5.3. Модифицированный алгоритм Джонсона. Контур выделяется, как только он обнаруживается в результате обхода графа. Для хранения текущего пути используется стек, используются также и списки $B(u)$.

Однако вершина v включается в список $B(u)$, если $(v, u) \in E$ и исследование этой дуги не ведет к выделению нового контура. В дополнение к этой структуре мы используем *позиционный вектор* \bar{p} и булевский *вектор достижимости* \bar{r} . Если вершина v есть j -я по счету вершина, считая от самой глубокой вершины в стеке, то будем говорить, что ее позиция $\bar{p}(v)$ равна j ; если вершина v удалена из стека, то будем говорить, что ее позиция равна $N + 1$; $\bar{p}(v) = N + 1$. Если вершина v еще не удалялась из стека, то полагают $\bar{r}(v) = ложь$, в противном случае $\bar{r}(v) = истина$. Вершина v блокируется (помечается), когда она вводится в стек, и эта метка сохраняется, пока вершина находится в стеке. При выводе из стека v разблокируется, только если был найден контур, проходящий через нее. Если вершина v покидает стек, имея метку, то она разблокируется, как только из стека будет удалена вершина z_1 при отыскании контура с начальной вершиной z_1 , причем найдется путь $[z_k, z_{k-1}, \dots, z_1]$, $z_k = v$, такой, что $z_{i+1} \in B(z_i)$, $k > i \geq 1$.

Алгоритм работает с графами, заданными списками смежности. Предварительно в графе отыскиваются бикомпоненты. В каждой бикомпоненте выбирается начальная вершина, в качестве которой берется вершина с наибольшей степенью захода. Начальная вершина удаляется из стека только после перечисления всех контуров в этой бикомпоненте, поэтому и требуется только одна начальная вершина на бикомпоненту.

Рассмотрим случаи, которые могут встретиться при удлинении текущего пути $[v_1, v_2, \dots, v_{k-1}]$ дугой (v_{k-1}, v_k) .

1. Если v_k не помечена, то, значит, она отсутствует в стеке, текущий путь удлиняется, вершина v_k заносится в стек и проверке подвергается дуга, исходящая из v_k .

2. Если v_k помечена и не находится в стеке, то она недоступна и должна быть пропущена. Вершина v_{k-1} заносится в список $B(v_k)$, а v_k удаляется из $A(v_{k-1})$.

3. Если v_k помечена и находится в стеке, то, значит, найден контур, который сразу же выписывается. Чтобы на этом шаге не порождались одинаковые контуры, используется наблюдение: порожденный контур есть новый контур тогда и только тогда, когда по крайней мере одна из его вершин ни разу не удалялась из стека. Тот факт, что вершина не удалялась из

стека, запоминается с помощью переменной q , локальной по отношению к рекурсивной процедуре. Для каждого данного вызова процедуры КОНТ переменная q обозначает самую верхнюю вершину в стеке, которая никогда не удалялась. Поэтому, если позиция $\bar{p}(v_k) \leq q$, то найден новый контур; в противном случае порожден дублетный контур и v_{k-1} вносится в список $B(v_k)$, а v_k удаляется из $A(v_{k-1})$.

В случаях 2 и 3, когда вершина v_k помечена, элементарный путь не удлиняется. Если простой путь не может быть больше удлинен, алгоритм переходит к предыдущей вершине в стеке, и т. д. Когда начальная вершина удаляется из стека, рассматривается новая бикомпонента, пока не будут рассмотрены все вершины.

А л г о р и т м:

В х о д: Граф $G = (V, E)$, заданный списками смежности $A(v)$.

В ы х о д: Список контуров.

начало

1. процедура КОНТ(целый v , q ; логический f);

2. **начало**

3. процедура НЕКОНТ(x , y);

4. **начало**

5. включить x в $B(y)$;

6. удалить y из $A(x)$;

7. **конец**;

8. процедура РАЗМАРК(x);

9. **начало**

10. $m(x) := \text{ложь}$;

11. для $y \in B(x)$ цикл

12. **начало**

13. включить x в $A(y)$;

14. если $m(y)$ то РАЗМАРК(y)

15. **конец** цикла;

16. положить $B(x) := \emptyset$;

17. **конец** РАЗМАРК;

18. $m(v) := \text{истина}$; $f := \text{ложь}$;

19. включить v в стек;

20. $t := \text{число вершин в стеке}$;

21. $\bar{p}(v) := t$;

22. если $\neg \neg r(v)$ то $q := t$;

23. для $w \in A(v)$ цикл

24. **начало**

25. если $\neg m(v)$ то

26. **начало**
27. $\text{КОНТ}(w, q; g);$
28. **если** g **то** $f := \text{истина}$
29. **иначе** $\text{НЕКОНТ}(v, w)$
30. **конец**
31. **иначе** **если** $\bar{p}(w) \leq q$ **то**
32. **начало**
33. вывести контур от w до v из стека, потом удалить w ;
34. $f := \text{истина};$
35. **конец**
36. **иначе** $\text{НЕКОНТ}(v, w);$
37. удалить v из стека;
38. **если** f **то** $\text{РАЗМАРК}(v);$
39. $\bar{r}(v) := \text{истина};$
40. $\bar{p}(v) := N + 1;$
41. **конец** цикла;
42. **конец** $\text{КОНТ};$
43. $A_D(v) := \text{списки смежности для бикомпоненты } D;$
44. **для** $j = 1$ **шаг** 1 **до** N **цикл**
45. $m(j) := \bar{r}(j) := \text{ложь};$
46. **для** каждой нетривиальной бикомпоненты **цикл**
47. **начало**
48. $s := \text{вершина с наибольшей полустепенью захода в этой бикомпоненте};$
49. $\text{КОНТ}(s,);$
50. **конец** цикла;
- конец**

Корректность алгоритма устанавливается последовательностью лемм, доказательства которых не приводятся.

Лемма 38. *Каждая вершина попадает в стек по крайней мере один раз.*

Лемма 39. *Если v_1, \dots, v_k — вершины, содержащиеся в данный момент времени в стеке и найден новый контур, проходящий через v_k , то вершины v_1, \dots, v_k оказываются непомянутыми после удаления их из стека.*

Лемма 40. *Пусть $[v_1, \dots, v_k, v_1]$ — контур такой, что v_1, \dots, v_k или циклическая перестановка их уже встречались в k верхних позициях стека в некоторый ранний момент времени и по крайней мере одна из этих вершин прежде удалялась из стека. Если v_1, \dots, v_k*

занимают теперь k верхних позиций в стеке, то они удаляются из стека.

Лемма 41. Пусть $\{z_1, \dots, z_k\}$ — простой путь, (z_k, v) — дуга, причем v находится в стеке и ранее из него не удалялась. Тогда, если z_1, \dots, z_k еще не в стеке, то z_1 не помечена.

Лемма 42. Пусть v_1, \dots, v_k, v_1 — подходящая циклическая перестановка для контура, такая, что вершина v_1 была первой среди $v_j, j \in [1:k]$, при каком-нибудь включении в стек. Тогда существует такая конфигурация стека, при которой вершины $v_1, \dots, v_j, j \in [1:k]$, встречаются в j верхних позициях стека прежде, чем v_1 покидает стек в первый раз.

Лемма 43. Если вершина находится в стеке, то она помечена.

Лемма 44. Каждый простой контур в графе порождается по крайней мере один раз.

Лемма 45. Каждый простой контур порождается самое большее один раз.

Трудоёмкость алгоритма составляет $O(m)$ на каждый простой контур, за исключением первого, на который тратится $O(n + m)$ операций.

5.4. Отыскание контуров фиксированной длины. Алгоритм использует матрицу смежности $A(G)$ и матрицы A^{k-1} и A^k , если длина отыскиваемых контуров равна k . Выберем некоторое i , такое, что $a_{ii}^{(k)} = 1$. Это означает, что вершина v_i принадлежит контуру длины k . Тогда вершина v_j принадлежит тому же контуру, если выполняются следующие три условия:

- а) $a_{jj}^{(k)} = 1$;
- б) $a_{ij} = 1$, т. е. существует путь длины 1 из v_i в v_j ;
- в) $a_{ji}^{(k-1)} = 1$, т. е. существует путь длины $k - 1$ из v_j в v_i .

Алгоритм основан на последовательной проверке условий до выявления всех вершин контура.

5.5. Цикломатической матрицей орграфа называется прямоугольная $(0-1)$ -матрица $H = \|h_{ij}\|$ размера $s \times n$, где s — число контуров, а n — число вершин и

$$h_{ij} = \begin{cases} 1, & \text{если вершина } v_j \text{ принадлежит контуру } C_i; \\ 0 & \text{в противном случае.} \end{cases}$$

Матрицей вложенности орграфа G называется матрица $E = \|e_{ij}\|$ размера $s \times s$, где s — число контуров, а

$$e_{ij} = \begin{cases} 1, & \text{если контур } C_i \text{ вложен в контур } C_j, \text{ т. е.} \\ & \text{все вершины контура } C_i \text{ принадлежат} \\ & \text{контуру } C_j; \\ 0 & \text{в противном случае.} \end{cases}$$

Построение матрицы вложенности происходит по правилу: если рассматривать строки цикломатической матрицы как булевские векторы \bar{C}_i и применять к ним теоретико-множественные операции, то достаточно положить:

а) $e_{ji} = 1$ и $e_{ij} = 0$, если $\bar{C}_i \cap \bar{C}_j = \bar{C}_i$ (контур C_i вложен в контур C_j);

б) $e_{ji} = 0$ и $e_{ij} = 1$, если $\bar{C}_i \cap \bar{C}_j = \bar{C}_j$ (контур C_j вложен в контур C_i);

в) $e_{ji} = 0$ и $e_{ij} = 0$, если $\bar{C}_i \cap \bar{C}_j \neq \bar{C}_i \neq \bar{C}_j$.

Полученную матрицу вложенности E можно рассматривать как матрицу смежности некоторого бесконтурного орграфа, в котором вершина $y \in \Gamma x$ для некоторой вершины x , если контур C_y вложен в контур C_x . Представление вложенности в виде орграфа позволяет в свою очередь организовать обход исходного графа в порядке, отражающем вложенность контуров, а также использовать его для сегментации программ.

Пример. Для графа с цикломатической матрицей, представленной на рис. 2.35, а, соответствующая ей матрица вложенности показана на рис. 2.35, б, а граф вложенности — на рис. 2.35, в.

5.6. Гамильтоновы контуры и циклы. Для нас интереснее к гамильтонову циклу связан в первую очередь с задачей о коммивояжере, заключающейся в отыскании во взвешенном графе гамильтонова цикла минимальной длины. К задаче о коммивояжере сводится большое число оптимизационных задач, однако ее ценность состоит еще в том, что она является своеобразным эталоном сложности. Существующие точные методы не позволяют решать ее уже для графов с числом вершин, приближающимся к ста. Известные приближенные методы дают удовлетворительное для практики решение для графов с числом вершин до 2000.

5.7. Эйлеров цикл соответствует обходу всех ребер графа, причем каждое ребро при таком обходе проходит в точности один раз и только в одном направлении, в то время как гамильтонов цикл соответствует обходу всех вершин.

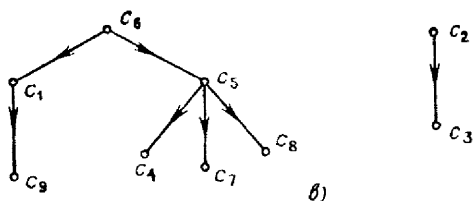
Теорема 17. *Связный неориентированный граф содержит эйлеров цикл тогда и только тогда, когда степень каждой вершины четна.*

		Номера вершин																							
		1 1 1 1 1 1 1 1 1 1										2 2 2 2 2 2 2 2 2 2													
		1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4
C_1		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
C_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
C_3		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C_4		0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C_5		0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
C_6		0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
C_7		0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C_8		0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C_9		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

а)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	1	1	0
6	1	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

б)



в)

Рис. 2.35.

Необходимость условия очевидна, так как эйлеров цикл, проходя по каждому ребру, должен столько раз выйти из вершины (каждый раз по новому ребру), сколько он в нее заходил.

Для доказательства достаточности дадим алгоритм построения эйлерова цикла в графе, удовлетворяющем условиям теоремы. Начиная с некоторой вершины v_0 , будем обходить ребра графа, стирая пройденные ребра

и образующиеся при этом изолированные вершины, пока не придем снова в вершину v_0 . Это возможно, так как, заходя в вершину по непройденному ранее ребру, мы имеем возможность выйти из него в силу четности степени вершины. Если при возвращении в v_0 все ребра окажутся пройденными, то нужный цикл построен. В противном случае в оставшейся непройденной части графа возьмем вершину v_1 , принадлежащую построенному циклу Φ и являющуюся началом еще не пройденного ребра. Начиная с v_1 , будем обходить не принадлежащие Φ ребра до тех пор, пока не придем снова в v_1 . Это возможно, поскольку в оставшейся после удаления ребер цикла Φ части графа все степени вершин четные. Построим новый цикл Φ' , объединяя циклы Φ и Φ_1 (вновь построенный) следующим образом: берем часть цикла Φ от v_0 до v_1 , цикл Φ_1 и затем оставшуюся часть цикла Φ от v_1 до v_0 . Если после этого в графе останутся непройденные ребра, процесс продолжаем, взяв цикл Φ' в качестве Φ . Через конечное число шагов процесс закончится построением эйлерова цикла, так как постоянно будут выполняться условия теоремы.

Теорема 18. *Связный орграф содержит эйлеров контур тогда и только тогда, когда для каждой вершины полустепень захода равна полустепени исхода.*

Следствие. *Каков бы ни был связный граф, всегда существует обход его ребер, при котором каждое ребро проходится дважды — в прямом и обратном направлениях.*

В самом деле, достаточно заменить каждое ребро парой противоположно направленных дуг, чтобы получить орграф, удовлетворяющий условиям теоремы 18.

5.8. Цикломатическим числом $\lambda(G)$ графа G называется величина $\lambda(G) = m(G) - n(G) + \kappa(G)$, где n — число вершин, m — число ребер, κ — число компонент связности.

Заметим, что цикломатическое число определено только для неориентированного графа.

Для связного графа $\kappa = 1$ и выражение для λ приобретает вид $\lambda = m - n + 1 = m - (n - 1)$, откуда немедленно следует, что λ есть число ребер, не принадлежащих произвольно выбранному каркасу. Так как каждое такое ребро, называемое *хордой* графа, присоединенное к каркасу, относительно которого определено

рассматриваемое множество хорд, порождает в точности один цикл, то мы получаем множество циклов, характеризующихся тем, что каждый из них имеет хотя бы одно ребро, не содержащееся в другом цикле этого множества. Назовем множество полученных таким образом циклов *множеством фундаментальных циклов (относительно фиксированного каркаса)*. Основное свойство множества фундаментальных циклов состоит в том, что оно является базисом пространства циклов графа, т. е. любой цикл может быть представлен как линейная комбинация фундаментальных циклов.

Множество фундаментальных циклов может быть аналогичным образом определено для орграфа, если под циклом понимать цикл для орграфа (т. е. замкнутую последовательность дуг), а под каркасом — некоторый оркаркас.

Подобно цикломатической матрице определяется *матрица фундаментальных циклов* C_f . Для ее построения перенумеруем фундаментальные циклы относительно каркаса T некоторым произвольным способом: $1, 2, \dots, \lambda(G) = m - n + 1$. Пусть хорда, которая появляется в цикле C_i , нумеруется как ребро i , $i \in [1 : \lambda(G)]$. Остальные ребра, т. е. ребра каркаса, занумеруем $\lambda + 1, \lambda + 2, \dots, m$. Тогда матрица фундаментальных циклов примет вид $C_f = [IC_{12}]$, где I — единичная матрица порядка λ . Так как C_f есть подматрица цикломатической матрицы данного графа, то ранг цикломатической матрицы не меньше ранга матрицы фундаментальных циклов, т. е. не меньше λ . Более того, можно доказать, что ранг цикломатической матрицы в точности равен λ , откуда и вытекает основное свойство множества фундаментальных циклов.

Связь матрицы C_f фундаментальных циклов с матрицей инцидентности $B(G)$ графа устанавливается теоремой, которую мы приводим без доказательства.

Теорема 19. *Матрица инцидентности B графа G может быть представлена в виде*

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

где B_{12} — неособенная квадратная матрица порядка $n - 1$. Матрица фундаментальных циклов для каркаса, соответствующего столбцам подматрицы B_{12} , имеет вид $C_f = [IB_{11}(B_{12}^{-1})^T]$.

5.9. Граф называется *контурно-связным* (*LC-граф*), если он содержит две вершины, скажем v_A и v_B , и три простых контура C_α , C_β и C_γ , такие, что:

- а) не существует вершины, одновременно принадлежащей всем трем контурам C_α , C_β и C_γ ;
- б) C_α содержит вершину v_A , но не содержит v_B ;
- в) C_β содержит вершину v_B , но не содержит v_A ;
- г) C_γ содержит и v_A , и v_B .

Пример. Граф на рис. 2.36 контурно-связен относительно вершин v_B и v_C .

Граф называется *k-контурно-связным* (*LC(k)-граф*), где $k \geq 2$, если в нем найдутся k вершин v_1, \dots, v_k , k простых контуров C_1, \dots, C_k и простой контур \bar{C} , такие, что:

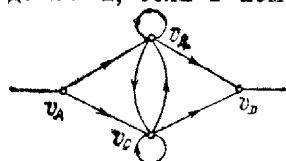


Рис. 2.36.

- а) не существует вершины, одновременно принадлежащей контурам \bar{C} и C_1, \dots, C_k ;
- б) для всех $i, i \in [1:k]$, контур C_i содержит все вершины v_1, \dots, v_k , кроме v_i ;
- в) \bar{C} содержит все вершины v_1, \dots, v_k .

Заметим, что контурно-связный граф есть 2-контурно-связный. Граф называется *KLC-графом*, если найдется такое k , что граф является k -контурно-связным.

Граф называется *петлевым*, если у каждой вершины есть петля и эти петли исчерпывают все множество контуров графа.

Назовем подмножество V' вершин орграфа $G = (V, E)$ *сечением*, если суграф $G' = (V - V', E')$, получаемый из G удалением вершин из V' вместе с инцидентными им дугами, является бесконтурным. Сечение называется *минимальным*, если оно содержит минимальное число вершин. Число вершин в минимальном сечении графа называется его *контурным рангом* k_G или просто *рангом*.

Библиографический комментарий

Параграф 1 написан по материалам работ И. В. Поттосина [69, 70], С. Эрнеста, К. Балке и Дж. Андерсона [110], В. Н. Касьянова [44—47]. Понятие прямой нумерации было введено в работе [110], аранжировки — в работах И. В. Поттосина, N -, L -, K -, T -нумерации — в работах В. Н. Касьянова.

Различным методам построения транзитивных замыканий посвящена обширная литература; один из первых алгоритмов,

118

по-видимому, принадлежит Дж. Бэйкеру [97]; также одним из первых, но наиболее часто применяемым является алгоритм С. Уоршалла [162] (см. также [61, 72]). Алгоритм Уоршалла послужил основой для создания целого ряда других алгоритмов. Среди них алгоритмы Л. Торелли [155], П. Пурдома [144] и А. Мунро [136]. Алгоритм Торелли имеет трудоемкость $O(n^4)$ и рассчитан на работу с графами, заданными списками смежности; алгоритм Пурдома имеет трудоемкость $O(n^3)$ при прочих равных условиях. Укажем еще на алгоритм Арлазарова — Диница — Кронрода — Фараджева [5] с трудоемкостью $O(n^3/\log n)$. Алгоритм замыкания относительно множества вершин принадлежит В. В. Мартынюку [60]; аналогичный алгоритм, но с трудоемкостью $O(n^2)$ был предложен ранее А. П. Ершовым в [29].

Простейший матричный алгоритм отыскания бикомпонент, описанный в § 2, принадлежит, по-видимому, С. Рамамурти; долгое время была надежда получить на его основе простой алгоритм отыскания иерархии зон, что оказалось неоправданным [146]. Несколько методов описано в книге А. А. Зыкова [33]; наиболее эффективные алгоритмы основаны на обходе графа, представленного списками смежности, с использованием стратегии поиска в глубину. К алгоритмам такого рода относятся описанный в п. 2.3 § 2 алгоритм И. А. Фараджева [76], а также алгоритмы А. В. Карзанова [39] и Р. Тарьяна [150]. Близок к ним (в смысле использования стратегии поиска в глубину) алгоритм В. Н. Касьянова [42], основанный на специальных нумерациях вершин. Алгоритмы отыскания вложенных зон взяты из работ [46] (алгоритм, основанный на K -нумерации), [92] (матричный алгоритм) и [110] (алгоритм на базе прямой нумерации). Понятие линейной компоненты принадлежит И. В. Поттосину [69], алгоритм отыскания линейных компонент принадлежит В. Н. Касьянову [46], алгоритм отыскания блоков — Е. А. Диницу, М. А. Зайцеву и А. В. Карзанову [24].

Понятие гамака было впервые сформулировано А. П. Ершовым, приведенное в книге определение принадлежит И. В. Поттосину [69], алгоритм отыскания гамаков — В. Н. Касьянову [43], понятия полугамака и шлейфа — В. Э. Иткину [37, 38]. Понятия интервала и сводимого графа восходят к работам Дж. Кока [49] и Ф. Аллен [93]; связь сводимости и аранжировки установлена В. Н. Касьяновым [41], ему же принадлежит теорема 12. Альтернативное определение сводимого графа, принадлежащее М. Хехту и Дж. Ульману, дано в [116]; алгоритм проверки графа на сводимость разработан Р. Тарьяном [152]; Дж. Хоукрофт и Дж. Ульман дали его улучшенный вариант в [118]. Длинные последовательности вершин в сводимых графах описаны в работах [90, 129]. Эффективный алгоритм объединения множеств может быть найден в [7], доказательство леммы 31 — в [117].

Одним из первых алгоритмов перечисления контуров является алгоритм Дж. Тьернана [156], его стратегия была использована в алгоритме А. Берзтисса [10] и др. Улучшенный алгоритм был предложен Х. Вайнблатом [163] (см. также [25]). Р. Тарьян [151] показал, что существуют графы, на которых алгоритм Вайнблата имеет экспоненциальную трудоемкость, и предложил другой метод с трудоемкостью $O(nm)$ на контур; такую же трудоемкость имеет алгоритм Арлазарова — Ускова — Фараджева [6]. Алгоритм с трудоемкостью $O(n + m)$ на кон-

тур был предложен Р. Ридом и Р. Тарьяном [145]. Алгоритм с такой же трудоемкостью, но основанный на оригинальной технике расстановки меток, был предложен Д. Джонсоном [119]; модифицированный алгоритм с той же трудоемкостью, продолжающий линию Тьернана — Тарьяна — Джонсона, был предложен Дж. Шварцфитером и П. Лаутером [149].

Дать краткий обзор литературы по методам решения задачи о нахождении гамильтонова цикла, достаточным условиям существования гамильтонова цикла и методам решения задачи о коммивояжере не представляется возможным, некоторая полезная информация может быть найдена в книгах [27, 33, 52, 53, 80].

ГЛАВА 3

ИТЕРАТИВНЫЕ АЛГОРИТМЫ ГЛОБАЛЬНОГО АНАЛИЗА ГРАФОВ. ПУТИ И ПОКРЫТИЯ

§ 1. Итеративный алгоритм Килдала

1.1. Полурешеткой называется множество L с идемпотентной, коммутативной и ассоциативной бинарной операцией. Обозначая результат этой операции над элементами a и b из L через $a \wedge b$ и используя в более сложных случаях скобки, мы получим, что полурешетка определяется следующими аксиомами:

а) $a \wedge a = a$;

б) $a \wedge b = b \wedge a$;

в) $a \wedge (b \wedge c) = (a \wedge b) \wedge c$

для всех $a, b, c \in L$.

Операция \wedge определяет частичный порядок на элементах из L по правилу: $x \leq y \Leftrightarrow x \wedge y = x$, $\forall x, y \in L$. Аналогично, $x < y \Leftrightarrow x \leq y \wedge x \neq y$. Для данного $X \equiv L$ запись $\bigwedge_{x \in X} x$ или $\bigwedge_{1 \leq i \leq n} x_i$ означает попарное приме-

нение операции \wedge (для простоты будем говорить о пересечении) к элементам из X . Предполагается, что L содержит нулевой элемент 0 , $0 \wedge x = 0$. Можно (но не обязательно) предполагать, что в L есть максимальный элемент или единица 1 , т. е. элемент, для которого $1 \wedge x = x$, $\forall x \in L$. Если в L единицы не было, то можно рассматривать пополненное множество L' , добавив к L единицу 1 .

Для данной полурешетки L допустимым множеством функций F называется множество функций f на L , удовлетворяющих следующим условиям:

1. Каждая функция $f \in F$ дистрибутивна относительно операции \wedge ; т. е. для любых x и y из L имеет место равенство $f(x \wedge y) = f(x) \wedge f(y)$.

2. Существует тождественная функция $e \in F$, т. е. для всех $x \in L$ имеем $e(x) = x$.

3. Множество F замкнуто относительно композиции функций, т. е. $[fg](x) = f(g(x)) \in F$, если $f \in F$ и $g \in F$.

4. Для каждого $x \in L$ существует конечное подмножество $H \subseteq F$, такое, что $x = \bigwedge_{f \in H} f(0)$.

Для данной полурешетки L последовательность $[x_1, \dots, x_n]$ элементов из L называется *цепью*, если для $i \in [1:n]$ имеем $x_i > x_{i+1}$. Полурешетка L называется *ограниченной*, если для каждого $x \in L$ существует константа b_x такая, что любая цепь, начинающаяся в x , имеет длину, не превосходящую b_x .

Пусть теперь L — ограниченная полурешетка, и пусть F — множество функций, определенных на L , причем множество F удовлетворяет условию 1 относительно L . Тогда для любого конечного подмножества $J \subseteq L$ имеет место соотношение $f\left(\bigwedge_{x \in J} x\right) = \bigwedge_{x \in J} f(x)$.

Лемма 1. Для данной ограниченной полурешетки L и множества F функций на L , удовлетворяющих условиям 1—3, F есть допустимое множество для L' , где

$$L' = \{x \mid \exists f_1, \dots, f_n \in F \text{ и } x = \bigwedge_{1 \leq i \leq n} f_i(0)\}.$$

Так как любое подмножество L , замкнутое относительно операции \wedge , удовлетворяет условиям идемпотентности, коммутативности и ассоциативности относительно \wedge и, следовательно, является полурешеткой, достаточно показать, что F удовлетворяет условиям 1—3 относительно L' , если F удовлетворяет условиям 1—3 относительно L .

Условия 1 и 2 удовлетворяются для F относительно L' , так как L' есть подмножество L . Предположим, что F не удовлетворяет условию 3 относительно L' , т. е. существуют $x \in L'$ и $f, g \in F$, для которых $fg(x) \notin L - L'$. Но из того, что $x \in L'$, следует $x \in \bigwedge_{h \in H} h(0)$

для некоторого конечного $H \subseteq F$. Таким образом, $fg(x) = fg\left(\bigwedge_{h \in H} h(0)\right) = \bigwedge_{h \in H} fgh(0)$. Следовательно, $fg(x)$ принадлежит L' по определению, что противоречит нашему предположению.

Лемма 2. Для данной ограниченной полурешетки L с сопоставленной ей системой функций F из $x \geq y$, где $x, y \in L$, следует, что $f(x) \geq f(y)$ для всех $f \in F$.

Лемма 8. Для любой ограниченной полурешетки L и любого счетного $J \cong L$ из неравенства $x \geq y$, выполняемого для всех $x \in J$, следует $\bigwedge_{x \in J} x \geq y$.

1.2. Пусть $G = (V, E)$ есть орграф с входом s . Пусть, далее, дана полурешетка L с операцией \wedge и множеством F сопоставленных ей функций, и пусть $M: V \rightarrow F$ есть отображение множества вершин V в множество F , ставящее в соответствие вершине v функцию $f_v(x)$ или, в другой записи, $f(v, x)$. Итеративный алгоритм глобального анализа орграфа G состоит в вычислении «меток» вершин при следующих условиях:

а) все метки принимают свои значения из полурешетки L ;

б) задана начальная метка (чаще всего это 0 полурешетки) для входной вершины s ;

в) метка в вершине v определяется метками ее непосредственных предшественников с помощью функций f_v .

Процедура расстановки меток носит название *разметки* вершин графа; в отличие от нумерации разметка допускает наличие у разных вершин одинаковых меток.

Алгоритм Килдала ($1 \notin L$):

Вход: Граф $G = (V, E)$ с входом s и N -номерами вершин (в алгоритме N -номера используются для идентификации вершин), полурешетка L и система функций F .

Выход: Граф G с размеченным множеством вершин.

Примечание. Если через $B(v)$ обозначается множество предшественников вершины v , т. е. $B(v) = \Gamma_v^{-1}$, то через $B^*(v)$ в алгоритме обозначается множество тех вершин из $B(v)$, N -номера которых меньше N -номера вершины v .

начало

t : элемент из L ;

\mathfrak{A} : массив $[1, \dots, k]$ элементов из L ;

j : целый; s : булевский;

1. $\mathfrak{A}[1] := 0$;

Шаг 1

2. для j от 2 до k шаг 1 цикл

3. **начало**

4. $\mathfrak{A}[j] := \bigwedge_{q \in B^*(j)} f_q(\mathfrak{A}[q])$;

5. **конец** цикла;
 6. $s := \text{истина}$;
 7. **пока** s **цикл**
 8. **начало**
 9. $s := \text{ложь}$;
 Шаг 2
 10. **для** j **от** 2 **до** k **шаг** 1 **цикл**
 11. **начало**
 12. $t := \bigwedge_{q \in B(j)} f_q(\mathfrak{A}[q])$;
 13. **если** $t \neq \mathfrak{A}[j]$ **то**
 14. **начало**
 15. $s := \text{истина}$;
 16. $\mathfrak{A}[j] := t$;
 17. **конец**
 18. **конец** цикла;
 19. **конец** цикла;
конец

Теорема 1. Для графа G , полурешетки L , отображения $M: V \rightarrow F$ и дерева поиска в глубину T метка $\mathfrak{A}(i)$ вершины v с N -номером i после k -й итерации алгоритма равна $\mathfrak{A}(i) = \bigwedge_{p \in \mathcal{P}^{(k)}(i)} f_p(0)$, где $\mathcal{P}^{(k)}(i) = \{p | p = \mu[s, i] \text{ и } p \text{ содержит самое большое } k-1 \text{ обратных дуг согласно } N\text{-нумерации, индуцированной деревом } T\}$.

Доказательство проводится индукцией по числу k полных итераций алгоритма.

Базис ($k=1$). Мы действуем по индукции по N -номерам j вершин G .

Базис ($j=1$). Очевидно, что тривиальный путь есть единственный элемент в $\mathcal{P}^{(1)}(1)$, так как каждая дуга, входящая в 1, есть обратная дуга. Таким образом, $\mathfrak{A}(1)$ должна быть равна $e(0) = 0$. Так как мы положили $\mathfrak{A}(1) := 0$ в первой итерации, то базис есть.

Шаг индукции ($j > 1$). Все пути из входа 1 в вершину j могут быть записаны в виде $\langle p, j \rangle$, где p есть путь из 1 в $q \in B^*(j)$, если они не содержат обратных дуг. По предположению индукции $\mathfrak{A}(q) = \bigwedge_{p \in \mathcal{P}^{(1)}(q)} f_p(0)$ для каждой вершины q из $B^*(j)$.

(Здесь и далее $f_p(\cdot) = f_{i_m}(f_{i_{m-1}}(\dots f_{i_1}(\cdot) \dots))$ для пути $p = [i_1, i_2, \dots, i_m, i_{m+1}]$ и $f_p = e$ для $m=0$.) Шаг 1

алгоритма гарантирует, что

$$\begin{aligned} \mathfrak{A}(j) &= \bigwedge_{q \in B^*(j)} f_q(\mathfrak{A}(q)) = \bigwedge_{q \in B^*(j)} \bigwedge_{p \in \mathcal{P}^{(1)}(q)} f_{\langle p, j \rangle}(0) = \\ &= \bigwedge_{q \in \mathcal{P}^{(1)}(j)} f_q(0). \end{aligned}$$

Это завершает индукцию по j для случая $k = 1$.

Шаг индукции ($k > 1$). Снова применим индукцию по j .

Базис ($j = 1$). Очевидно, что $0 = \bigwedge_{p \in \mathcal{P}^{(1)}(1)} f_p(0) \geq \geq \bigwedge_{p \in \mathcal{P}^{(k)}(1)} f_p(0)$. Таким образом, $\bigwedge_{p \in \mathcal{P}^{(h)}(1)} f_p(0) = 0$.

После k итераций алгоритма $\mathfrak{A}(1)$ остается равной 0, так что $\mathfrak{A}(1) = \bigwedge_{p \in \mathcal{P}^{(h)}(1)} f_p(0)$ после k -й итерации.

Шаг индукции ($j > 1$). Каждый путь Q в $\mathcal{P}^{(k)}(j)$ имеет форму $\langle p, q \rangle$, где либо 1) $q \in B^*(j)$ и $p \in \mathcal{P}^{(k)}(q)$, либо 2) $q \in B(j) \setminus B^*(j)$ и $p \in \mathcal{P}^{(k-1)}(q)$. Другими словами, 2) соответствует случаю, когда $\langle q, j \rangle$ есть обратная дуга, т. е. $q \geq j$, а 1) соответствует обратному случаю, когда $q < j$. Когда в алгоритме переменная t вычислена для j при k -й итерации, метка $\mathfrak{A}(q)$ будет равна:

а) $\bigwedge_{p \in \mathcal{P}^{(k)}(q)} f_p(0)$, если $q < j$, т. е. в случае 1), примененном к q ;

б) $\bigwedge_{p \in \mathcal{P}^{(k-1)}(q)} f_p(0)$, если $q \geq j$, т. е. в случае 2).

Заметим, что а) следует из предположения индукции для j , а б) — из предположения индукции для k .

На k -й итерации переменная t полагается равной $\bigwedge_{q \in \mathcal{P}^{(k)}(j)} f_q(0)$, и, таким образом, метке $\mathfrak{A}(j)$ прида-

ется это значение, если оно не было дано ей раньше.

Лемма 4. Алгоритм останавливается после k итераций тогда и только тогда, когда для каждой вершины $j \in V$ и для каждого пути $p \in \mathcal{P}(j)$ существуют пути p_1, \dots, p_r , принадлежащие $\mathcal{P}^{(k-1)}(j)$ и такие, что $f_p(0) \geq \bigwedge_{1 \leq j \leq r} f_{p_j}(0)$.

Доказательство. Необходимость. Из леммы 3 имеем

$$\bigwedge_{q \in \mathcal{P}^{(k)}(j)} f_q(0) = \bigwedge_{q \in \mathcal{P}^{(k-1)}(j)} f_q(0),$$

так что равенство $t = \mathfrak{A}(j)$ на k -й итерации алгоритма обязательно имеет место, а переменная s будет иметь в конце этой итерации значение *ложь*.

Достаточность. Предположим, что алгоритм закончил свою работу после m -й итерации, $m \leq k$. Тогда для произвольной вершины j и пути $p \in \mathcal{P}(j)$ должно быть выполнено условие $f_p(0) \geq \bigwedge_{Q \in \mathcal{P}^{(m-1)}(j)} f_Q(0)$.

Мы должны показать, что существует конечное подмножество S множества $\mathcal{P}^{(m-1)}(j)$, такое, что $f_p(0) \geq \bigwedge_{Q \in S} f_Q(0)$. Пусть Q_1, Q_2, \dots — пути из $\mathcal{P}^{(m-1)}(j)$, перенумерованные в произвольном порядке, и пусть $x_i = \bigwedge_{1 \leq h \leq i} f_{Q_h}(0)$. Очевидно, что $x_i \geq x_{i+1}$ для всех i . Из условия ограниченности полурешетки вытекает существование только конечного числа значений i , для которых $x_i > x_{i+1}$. Пусть i_0 — наименьшее из таких значений, и пусть $S = \{Q_1, \dots, Q_{i_0}\}$. Тогда

$$\bigwedge_{Q \in \mathcal{P}^{(m-1)}(j)} f_Q(0) = \bigwedge_{Q \in S} f_Q(0),$$

так что S есть искомое конечное множество.

Теорема 2. Для данных полурешетки L , системы функций F , графа G и отображения $M: V \rightarrow F$ алгоритм Килдала останавливается после $d(G) + 3$ итераций, где $d(G)$ — наибольшее число обратных дуг в простых путях G , в том и только том случае, если система $D = (L, \wedge, F)$ удовлетворяет условию

$$(\forall f, g \in F) (\forall x \in L) [fg(0) \geq g(0) \wedge f(x) \wedge x]. \quad (3.1)$$

Доказательство. Необходимость. Исходя из леммы 4, достаточно показать, что для каждой вершины $j \in V$ и каждого пути $p \in \mathcal{P}(j)$ существуют пути p_1, \dots, p_m в $\mathcal{P}^{(d+2)}(j)$, такие, что $f_p(0) \geq \bigwedge_{1 \leq i \leq m} f_{p_i}(0)$. Мы докажем это утверждение индукцией по числу k обратных дуг, содержащихся в p , $p = [i_1, \dots, i_r]$, где $i_1 = 1$ и $i_r = j$.

Базис ($0 \leq k \leq d+1$). Этот случай тривиален, так как достаточно положить $m = 1$ и $p_1 = p$.

Индуктивный шаг ($k > d+1$). Так как путь p содержит больше чем $d+1$ обратных дуг, он не может не содержать контура по определению числа d . Возьмем наибольший номер a , такой, что $i_a = i_b$ для некоторого $b > a$. Мы утверждаем, что путь $p_1 =$

$= [i_1, \dots, i_a]$ должен содержать по крайней мере одну обратную дугу, ибо путь $[i_{a+1}, \dots, i_r]$ не содержит контуров и, таким образом, имеет самое большее d обратных дуг. Дуга (i_a, i_{a+1}) должна быть обратной, а так как путь p по предположению имеет более $d + 1$ обратных дуг, то наше утверждение справедливо. Путь $p_2 = [i_a, \dots, i_b]$ содержит по крайней мере одну обратную дугу, так как каждый контур в графе обязательно имеет одну обратную дугу относительно некоторого дерева поиска в глубину. Пусть теперь $p_3 = [i_b, \dots, i_r]$, а p_4 — путь без обратных дуг из вершины 1 в вершину i_a , как это показано на рис. 3.1. Такой путь по дереву поиска в глубину всегда существует.

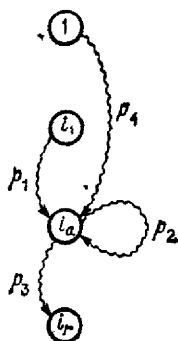


Рис. 3.1.

Положим теперь $x = f_{p_4}(0)$. Имеем

$$\begin{aligned} f_p(0) &= f_{p_3}(f_{p_2}(f_{p_1}(0))) \geq && \text{(по определению)} \\ &\geq f_{p_3}(f_{p_1}(0) \wedge f_{p_2}(x) \wedge x) = && \text{(по предположению)} \\ &= f_{p_3}(f_{p_1}(0) \wedge f_{p_2}(f_{p_4}(0)) \wedge f_{p_4}(0)) = \\ &= f_{p_3}f_{p_1}(0) \wedge f_{p_3}f_{p_2}f_{p_4}(0) \wedge f_{p_2}f_{p_4}(0) = \\ &= f_{p'}(0) \wedge f_{p''}(0) \wedge f_{p'''}(0), \end{aligned}$$

где $p' = [i_1, \dots, i_a, i_{b+1}, \dots, i_r]$, $p'' = \langle p_4, i_{a+1}, \dots, i_b, i_{b+1}, \dots, i_r \rangle$ и $p''' = \langle p_4, i_{b+1}, \dots, i_r \rangle$. p' , p'' и p''' — пути в G , причем каждый из них содержит самое большее $k - 1$ обратных дуг. Отсюда немедленно следует справедливость индуктивного предположения.

Достаточность. Предположим, что условие (3.1) не выполнено, т. е. имеет место соотношение

$$(\exists x \in L) (\exists f, g \in F) [fg(0) \geq g(0) \wedge f(x) \wedge x].$$

По условию 4), которому удовлетворяет система функций F , будучи допустимой для L , существуют функции $h_1, \dots, h_n \in F$ такие, что $x = \bigwedge_{1 \leq i \leq n} h_i(0)$. Возможны два случая.

Случай 1. $ff(x) \geq f(x) \wedge x$. На рис. 3.2 показан граф, для которого $d(G) = 0$. После третьей итерации алгоритма Килдала имеем $\mathfrak{A}(n+2) = x \wedge f(x) \wedge ff(x)$, в то время как $\mathfrak{A}(n+2) = x \wedge f(x)$ после второй итера-

ции. Следовательно, алгоритм требует не менее $4 > (d(G) + 3) = 3$ итераций.

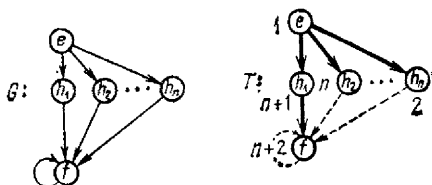


Рис. 3.2.

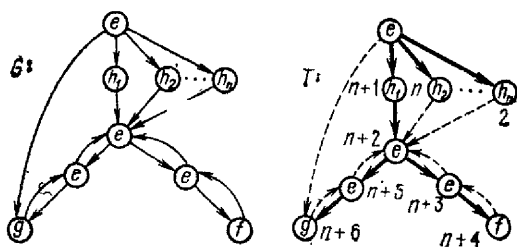


Рис. 3.3.

Случай 2. $ff(x) \geq f(x) \wedge x$. Рассмотрим граф на рис. 3.3 с $d(G) = 2$. Имеем:

после первой итерации: после второй итерации:

$$\mathfrak{A}(n+2) = x,$$

$$\mathfrak{A}(n+2) = x,$$

$$\mathfrak{A}(n+3) = x,$$

$$\mathfrak{A}(n+3) = x \wedge f(x),$$

$$\mathfrak{A}(n+4) = x,$$

$$\mathfrak{A}(n+4) = x \wedge f(x),$$

$$\mathfrak{A}(n+5) = x,$$

$$\mathfrak{A}(n+5) = x \wedge g(0),$$

$$\mathfrak{A}(n+6) = 0;$$

$$\mathfrak{A}(n+6) = 0;$$

после третьей итерации:

$$\mathfrak{A}(n+2) = g(0) \wedge f(x) \wedge x,$$

$$\begin{aligned} \mathfrak{A}(n+3) &= g(0) \wedge f(x) \wedge x \wedge ff(x) = \\ &= g(0) \wedge f(x) \wedge x, \end{aligned}$$

$$\mathfrak{A}(n+4) = g(0) \wedge f(x) \wedge x,$$

$$\mathfrak{A}(n+5) = g(0) \wedge f(x) \wedge x,$$

$$\mathfrak{A}(n+6) = 0;$$

после четвертой итерации:

$$\mathfrak{A}(n+2) = g(0) \wedge f(x) \wedge x,$$

$$\mathfrak{A}(n+3) = g(0) \wedge f(x) \wedge x \wedge fg(0),$$

$$\mathfrak{A}(n+4) = g(0) \wedge f(x) \wedge x \wedge fg(0),$$

$$\mathfrak{A}(n+5) = g(0) \wedge f(\bar{x}) \wedge x_n$$

$$\mathfrak{A}(n+6) = 0;$$

после пятой итерации:

$$\mathfrak{A}(n+2) = g(0) \wedge f(x) \wedge x \wedge fg(0),$$

$$\mathfrak{A}(n+3) = g(0) \wedge f(x) \wedge x \wedge fg(0) \wedge ffg(0),$$

$$\mathfrak{A}(n+4) = g(0) \wedge f(x) \wedge x \wedge fg(0) \wedge ffg(0) = \\ = g(0) \wedge f(x) \wedge x \wedge fg(0),$$

$$\mathfrak{A}(n+5) = g(0) \wedge f(x) \wedge x \wedge fg(0),$$

$$\mathfrak{A}(n+6) = 0.$$

Имеем $\mathfrak{A}(n+2) = g(0) \wedge f(x) \wedge x$ после четвертой итерации и $\mathfrak{A}(n+2) = g(0) \wedge f(x) \wedge x \wedge fg(0)$ после пятой итерации. Из предположения, что (3.4) не имеет места для f, g и x , вытекает, что $g(0) \wedge f(x) \wedge x \neq g(0) \wedge f(x) \wedge x \wedge fg(0)$. Таким образом, алгоритм Килдала требует не менее $6 > (d(G) + 3) = 5$ итераций

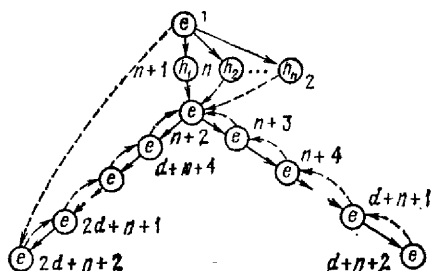


Рис. 3.4.

и, фактически, останавливается после шестой итерации. В общем случае любой граф, дерево поиска в глубину которого имеет вид, показанный на рис. 3.4, требует не менее $d(G) + 3$ итераций алгоритма. Теорема доказана.

1.3. Алгоритм Килдала для полурешетки с единицей. Скорость работы алгоритма выше, когда полурешетка L содержит единицу 1. Изменения в алгоритме незначительны.

начало

1. $\mathfrak{A}(1) := 0$;

2. для j от 2 до n шаг 1 цикл

3. начало

4. $\mathfrak{A}(j) := 1$;

5. $c := \text{истина}$;

6. конец цикла;

7. пока c цикл

8. начало

9. $c := \text{ложь}$;

10. для j от 2 до n шаг 1 цикл

11. начало

12. $t := \bigwedge_{q \in B(j)} f_q(\mathfrak{A}(q))$;

13. если $t \neq \mathfrak{A}(j)$ то

14. $c := \text{истина}$;

15. $\mathfrak{A}(j) := t$;

16. конец

17. конец цикла;

18. конец цикла;

конец

§ 2. Пути в орграфах

Простейшим примером применения разметки графов служат алгоритмы отыскания кратчайших путей в графе.

2.1. Пусть G — обыкновенный орграф, т. е. орграф, вершинам и дугам которого не приписаны никакие веса. Путь $\mu[a, b]$ называется *кратчайшим путем* из a в b , если длина его наименьшая из длин всех путей из a в b .

Пусть для простоты ищется кратчайший путь из входа графа s в выход t . Пусть, далее, L — множество целых положительных чисел, пополненное символом ∞ , операция \bigwedge есть взятие минимума, функция $f_v(\mathfrak{A}(v))$ есть «метка $\mathfrak{A}(v)$ плюс 1». Тогда алгоритм Килдала для случая, когда 1 принадлежит L , примет следующий вид. (Мы по-прежнему считаем, что все вершины имеют N -номера, $N(s) = 1$, $N(t) = n$.)

начало

1. $\mathfrak{A}(1) := 0$;

примечание роль нуля 0 в нашем случае играет число нуль **конец примечания**

2. для j от 2 до n шаг 1 цикл

3. $\mathfrak{A}(j) := \infty$;

примечание роль единицы 1 в нашем случае играет ∞ **конец примечания**

4. $c := \text{истина}$;

5. пока c цикл

6. **начало**

7. $c := \text{ложь}$;

8. для j от 2 до n шаг 1 цикл

9. **начало**

10. $t := \min_{q \in B(j)} \{\mathfrak{A}(q) + 1\}$;

11. если $t \neq \mathfrak{A}(j)$ то

12. **начало**

13. $c := \text{истина}$;

14. $\mathfrak{A}(j) := t$;

15. **конец**

16. **конец цикла**;

17. **конец цикла**;

конец

Однако для отыскания кратчайшего пути метод поиска в глубину не дает нужного порядка обработки вершин. Так, для графа на рис. 3.3 окончательная метка для вершины $n+2$ будет найдена после двух итераций алгоритма, т. е. вначале этой вершине будет приписана метка на основе метки вершины $n+1$, а затем уже будут учтены полученные позже метки вершин 2, 3, ..., n . Более подходящим является просмотр вершин, основанный на поиске в ширину, который приводит к традиционному виду алгоритма.

Шаг 1. Пометим вершину x_0 меткой $\lambda_0 = 0$.

Шаг 2. Все вершины $y \in \Gamma x_0$ пометим меткой $\lambda(y) = 1$.

Итерационный шаг. Пусть $V(k)$ — множество вершин, помеченных меткой k . Пометим вершины из множества $V(k+1) = \bigcup_{v \in V(k)} \Gamma v$, которые не были помечены ранее, меткой $\lambda = k+1$.

Алгоритм заканчивается, когда будет помечена искомая вершина x_n . Нетрудно убедиться, что метка $\lambda(y)$ вершины y есть длина кратчайшего пути из x_0 в y .

Собственно кратчайший путь из x_0 в x_n отыскивается обратным ходом от x_n к x_0 с общим шагом.

Итерационный шаг обратного хода. Если y_r — последняя найденная вершина кратчайшего пути, то в качестве вершины y_{r-1} берется вершина x такая, что $x \in \Gamma^{-1}y_r$ и $\lambda(x) = \lambda(y_r) - 1$.

Отыскание кратчайшего пути заканчивается, когда будет достигнута начальная вершина x_0 .

Алгоритм носит название *волнового алгоритма*.

2.2. Алгоритм Дейкстры. Пусть граф G не имеет контуров отрицательной длины. Для простоты будем считать, что все дуги u имеют положительные веса — длины $l(u)$.

В алгоритме Дейкстры выделяются две части: отыскание длины кратчайшего пути методом расстановки меток и отыскание собственно кратчайшего пути обратным ходом. Для удобства изложения будем считать, что ищется путь из входа орграфа x_0 в вершину x_n .

начало

1. $\lambda(x_0) := 0$;

2. объявить метку $\lambda(x_0)$ постоянной меткой;

3. всем вершинам $y \in \Gamma x_0$ приписать временные метки $\lambda(y) = l(x_0, y)$;

4. $x_1 :=$ вершина графа с наименьшей временной меткой;

5. объявить $\lambda(x_1)$ постоянной меткой;

6. для i от 1 до n шаг 1 цикл

7. **начало**

8. для всех $y \in \Gamma x_i$ цикл

9. **начало**

10. если y не имеет временной метки то

11. $\lambda(y) := \lambda(x_i) + l(x_i, y)$

12. иначе $\lambda(y) := \min \{\lambda(y), \lambda(x_i) + l(x_i, y)\}$;

13. **конец цикла**;

14. $x_{i+1} :=$ вершина графа с наименьшей временной меткой;

15. объявить метку $\lambda(x_{i+1})$ постоянной меткой;

16. **конец цикла**;

конец

Для отыскания собственно кратчайшего пути используется итерационный шаг обратного хода. Пусть y_r — последняя найденная вершина кратчайшего пути (вначале это — вершина x_n). В качестве вершины y_{r-1}

берется вершина x , такая, что $x \in \Gamma^{-1}y_r$ и $\lambda(x) = \lambda(y_r) - l(x, y_r)$.

Алгоритм заканчивается, когда будет достигнута вершина x_0 .

Пример. Пусть в графе на рис. 3.5 требуется найти кратчайший путь из a в b . Длины дуг обозначены числами возле соответствующей дуги. Последовательное изменение меток вершин показано в таблице 3.1. Окончательное распределение меток и кратчайший путь показаны на рис. 3.6.

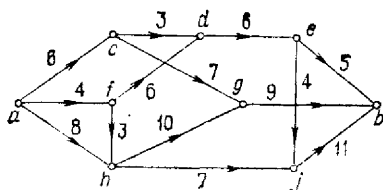


Рис. 3.5.

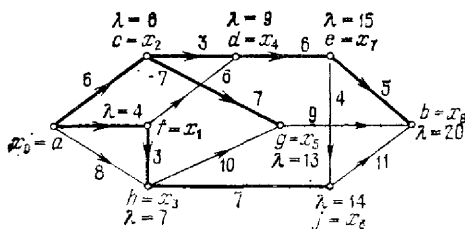


Рис. 3.6.

2.3. Критическим путем в орграфе с входом s и выходом t называется самый длинный путь между этими вершинами. Если кратчайший путь в орграфе не существует при наличии контуров отрицательной длины, то критический путь не существует в орграфе с контурами. В обоих случаях это соответствует неограниченности полурешетки L относительно операции \min в первом случае и операции \max во втором.

Итак, пусть G — бесконтурный граф. Отыскание критического пути в нем из входа s в выход t может быть осуществлено с помощью алгоритма кратчайшего пути, если веса $w(e)$ дуг заменить либо на $(-w(e))$, либо на $1/w(e)$. В последнем случае метки уже не должны рассматриваться как длины.

2.4. Отыскание кратчайшего пути в орграфе с контурами отрицательной длины. Кратчайшим путем между вершинами a и b в орграфе с контурами отрицательной длины принято считать *простой путь наименьшей длины* между a и b . Это приводит к мысли перечислить все простые пути между заданными вершинами и из них выбрать путь наименьшей длины.

Таблица 3.1

Вершина	Метки								
$a(=x_0)$	$\boxed{0}$	0	0	0	0	0	0	0	0
$c(=x_2)$	6	$\boxed{6}$	6	6	6	6	6	6	6
$f(=x_1)$	$\boxed{4}$	4	4	4	4	4	4	4	4
$h(=x_3)$	8	7	$\boxed{7}$	7	7	7	7	7	7
$d(=x_4)$		10	9	$\boxed{9}$	9	9	9	9	9
$g(=x_5)$			13	13	$\boxed{13}$	13	13	13	13
$j(=x_6)$				14	14	$\boxed{14}$	14	14	14
$e(=x_7)$					15	15	$\boxed{15}$	15	15
$b(=x_8)$						22	20	$\boxed{20}$	

Вначале рассмотрим алгоритм перечисления всех простых $s-t$ -путей в обыкновенном орграфе с использованием оператора склейки, допускающий обобщение на случай перечисления всех (!) путей в орграфе и легко реализующийся на ЭВМ.

Пусть орграф $G=(V, U)$ задан списком дуг:

$$G = \{e_m | e_m = \langle u, x, y \rangle, u = (x, y), u \in U\}.$$

Работа алгоритма состоит в последовательном удалении вершин (исключая s и t) и одновременной замене дуг их склейками — отрезками путей из s в t . Склейка дуг и отрезков путей производится оператором склейки Δ , который заменяет список путей из $\{x_i\}$ в вершину x — список S^+ — и список S^- путей из x в $\{y_j\}$ одним списком путей $S^+ \Delta S^-$ из $\{x_i\}$ в $\{y_j\}$.

Пусть $S^+ = \{s_1, \dots, s_N\}$, $s_i = \langle \mu_i, x_i, x \rangle$, $i = 1, \dots, N$, $S^- = \{t_1, \dots, t_M\}$, $t_j = \langle \mu_j, x, y_j \rangle$, $j = 1, \dots, M$, $S^+, S^- \neq \emptyset$, — указанные выше списки путей. Тогда склейкой $S^+ \Delta S^-$ путей из S^+ и S^- называется множе-

ство R простых путей, такое, что

$$R = \{r_{ij} \mid r_{ij} = \langle \mu[x_i, y_j], x_i, y_j \rangle, \mu[x_i, y_j] = \\ = \mu[x_i, x] \cup \mu[x, y_j], i = 1, \dots, N; j = 1, \dots, M \},$$

причем одинаковые элементы в R учитываются по одному разу.

З а м е ч а н и е. Употребление здесь теоретико-множественной операции объединения возможно, так как в силу принятых в главе 1 соглашений путь полностью определяется множеством своих вершин.

А л г о р и т м:

начало

1. $R_1 := U$;

примечание в качестве начального значения множества R путей из s в t берется множество дуг U
конец примечания

2. $X_1 := V \setminus \{s, t\}$;

3. **пока** X_1 не пусто **цикл**

4. **начало**

5. выбрать произвольную вершину x в X_1 ;

6. $X_1 := X_1 \setminus \{x\}$;

7. $R_1^+ = \{r \mid r = \langle \mu(r), x(r), x \rangle, r \in R_1, x(r) \in X_1 \cup \{s, t\}\}$;

примечание формируется множество путей, входящих в x **конец примечания**

8. $R_1^- = \{r \mid r = \langle \mu(r), x, y(r) \rangle, r \in R_1, y(r) \in X_1 \cup \{s, t\}\}$;

примечание формируется множество путей, исходящих из x **конец примечания**

9. $R_1^0 = R_1 \setminus (R_1^+ \cup R_1^-)$;

примечание формируется множество путей, не затрагиваемых удалением вершины x **конец примечания**

10. $R_1 := R_1^0 \cup (R_1^+ \Delta R_1^-)$;

11. **конец цикла**;

конец

Множество R_1 после окончания работы алгоритма даст искомое множество.

В случае взвешенного графа рассмотрим алгоритм с трудоемкостью $O(n + m)$ операций на путь, использующий метод разметки из алгоритма Джонсона перечисления контуров в орграфе. Алгоритм наряду с генерированием $s-t$ -путей вычисляет и их длину.

Пусть оргграф $G = (V, E)$ задан списками смежности, и пусть $w(e)$ — вес дуги $e = (\alpha(e), \beta(e))$, где через $\alpha(e)$ мы будем обозначать начало дуги, а через $\beta(e)$ — ее конец. Будем предполагать, что граф не имеет петель.

процедура ГЕНЕРИРОВАНИЕ ПУТЕЙ;

1. начало

$A(n), B(n)$: списочный массив; $\beta(e), w(e)$: массив;

b : логический массив;

2. процедура ОБХОД(v, f);

примечание v — есть последняя достигнутая при обходе вершина, переменная f имеет тип логический
конец примечания

3. начало

g : логический;

4. процедура РАЗБЛОК(u);

5. начало

6. $b(u) := \text{ложь}$;

7. для $y \in B(u)$ цикл

8. начало

9. удалить y из $B(u)$;

10. если $b(y)$ то РАЗБЛОК(y);

11. конец цикла;

12. конец РАЗБЛОК;

13. $f := \text{ложь}$;

14. $b(v) := \text{истина}$;

15. для $e \in A(v)$ цикл

16. начало

17. $y := \beta(e)$;

18. занести e в текущий список PS ;

19. $d := d + w(e)$;

20. если $y = t$ то

21. начало

22. занести путь, хранящийся в текущем PS , в список путей;

23. выдать длину пути из s в t ;

24. $f := \text{истина}$;

25. конец

26. иначе если $\neg b(y)$ то

27. начало

28. ОБХОД(y, g);

29. если g то $f := \text{истина}$;

30. конец

31. удалить e из текущего списка PS ;

32. $d := d - w(e)$;
 33. **конец цикла**;
 34. **если** f **то** РАЗБЛОК(v);
 35. **иначе** для $e \in \bar{A}(v)$ **цикл**
 36. **начало**
 37. $y := \beta(e)$;
 38. **если** $v \notin B(y)$ **то** занести v в $B(y)$;
 39. **конец цикла**;
 40. **конец ОБХОД**;
 41. $PS := \emptyset$;
 42. $d := 0$;
 43. для каждой вершины u графа G **цикл**
 44. **начало**
 45. $b(u) := \text{ложь}$;
 46. $B(u) := \emptyset$;
 47. **конец цикла**;
 48. ОБХОД(s ,);
- конец ГЕНЕРИРОВАНИЕ ПУТЕЙ**

Нетрудно убедиться, что этот алгоритм порождает все пути из s в t без дублирования и пропусков. Трудоемкость алгоритма составляет $O(n + m)$ на путь, требуемая память — $O(n + m)$.

2.5. Перечисление всех путей в графе. Рассмотрим некоторые свойства матриц смежности орграфов.

Лемма 5. Пусть $G = (X, U)$ и $H = (X, \hat{U})$ — два орграфа с одним и тем же множеством вершин и с матрицами смежности $A(G) = \|a_{ij}\|$ и $A(H) = \|b_{ij}\|$ соответственно. Матрице $A(G) \times A(H)$ — произведению матриц $A(G)$ и $A(H)$ — отвечает мультиграф, образованный следующим образом: из вершины x_i в вершину x_j идет столько дуг, сколько существует различных путей, ведущих из x_i в x_j и составленных из двух дуг, первая из которых принадлежит U , а вторая — \hat{U} .

Доказательство. Путь вида $[x_i, x_k, x_j]$, где $(x_i, x_k) \in U$, а $(x_k, x_j) \in \hat{U}$, существует, если $a_{ik} \cdot b_{kj} = 1$; поэтому общее число путей из x_i в x_j , образованных дугой из множества \hat{U} и последующей дугой из множества U , равно $\sum_{k=1}^n a_{ik} \cdot b_{kj}$, а это и есть (i, j) -й элемент матрицы $A(G) \times A(H)$.

Следствие 1. Элемент p_{ij} матрицы $P = A^\theta$, полученной возведением матрицы смежности $A(G)$ графа G в степень θ , равен числу различных путей длины θ , идущих из x_i в x_j .

Следствие 2. В графе G существует путь длины θ тогда и только тогда, когда $A^\theta \neq 0$; не существует контуров тогда и только тогда, когда $A^0 = 0$, начиная с некоторого θ_0 .

Метод перечисления всех путей в орграфе основан на лемме 5. Пусть $\mu_1 = [x_{i_1}, \dots, x_{i_p}, x_k]$ и $\mu_2 = [x_l, x_{j_1}, \dots, x_{j_q}]$ — некоторые простые пути. По определению оператора склейки Δ

$$\mu_1 \Delta \mu_2 = \begin{cases} [x_{i_1}, \dots, x_{i_p}, x_k, x_{j_1}, \dots, x_{j_q}], & \text{если } x_k = x_l \text{ и} \\ & \text{получившийся путь простой;} \\ \emptyset & \text{(пустая последовательность вершин) в про-} \\ & \text{тивном случае.} \end{cases}$$

Если $M_{x_i x_k}^p = \{\mu_1, \mu_2, \dots, \mu_r, \dots, \mu_R\}$ есть множество путей длины p из x_i в x_k , а $M_{x_k x_j}^q = \{\nu_1, \nu_2, \dots, \nu_l, \dots, \nu_T\}$ есть множество путей длины q из x_k в x_j , то $M_{x_i x_j}^p \Delta M_{x_k x_j}^q$ есть множество простых путей из x_i в x_j .

Для любых целых положительных чисел r и s имеем $M_{x_i x_j}^{(r+s)} = \bigcup_{k=1}^n (M_{x_i x_k}^r \Delta M_{x_k x_j}^s)$, $n = |X|$. Исходя из указанных соотношений, видим, что это есть не что иное, как определение «умножения» матриц с буквенными элементами $\|M\|^{(r+s)} = \|M\|^{(r)} \Delta \|M\|^{(s)}$. Беря в качестве $\|M\| = \|m_{ij}\|$ матрицу смежности $A(G)$, записываемую в виде

$$m_{ij} = \begin{cases} x_i x_j, & \text{если } a_{ij} = 1; \\ \emptyset & \text{в противном случае,} \end{cases}$$

нетрудно убедиться в том, что $\|M\|^{(1)}$ есть матрица путей длины 1, $\|M\|^{(2)} = \|M\|^{(1)} \Delta \|M\|^{(1)}$ — матрица путей длины 2 и т. д.

§ 3. Пути, удовлетворяющие дополнительным ограничениям. Покрытия

Задача о кратчайшем пути очень часто встречается в ситуации, когда необходимо учитывать дополнительные ограничения. Наличие их может значительно повысить сложность задачи.

3.1. Кратчайший путь, проходящий через заданное множество вершин. Рассмотрим два ограничения: кратчайший путь должен проходить через выделенное

множество вершин, и кратчайший путь должен содержать как можно меньше невыделенных вершин. Первая задача хорошо известна в теории исследования операций. Эффективно она решается только для бесконтурных графов. Рассмотрим ее для бесконтурного взвешенного графа. С одной стороны, этот случай включает в себя и обыкновенные графы, с другой стороны, он более близок к требованиям практики.

Итак, пусть G — бесконтурный орграф с множеством вершин V . Пусть, далее, C есть подмножество V ; каждой вершине v_i из C приписан вес $w(v_i) > 0$. Требуется отыскать такой $s-t$ -путь, чтобы сумма весов входящих в него вершин из C была максимальна. Предлагаемый алгоритм основан на сведении задачи к задаче о критическом пути и имеет трудоемкость $O(m)$, где m — число дуг. Определим для каждой дуги (x, y) в G длину $l(x, y)$, положив $l(x, y) = w(x)$, если $x \in C$, и $l(x, y) = 0$ в противном случае. Обозначим новый граф с взвешенными дугами через H .

Лемма 6. *Путь μ из s в t в графе G имеет наибольший суммарный вес вершин тогда и только тогда, когда μ имеет наибольшую длину среди всех $s-t$ -путей в графе H .*

В случае графа с контурами в качестве приближенного решения можно взять решение задачи для графа Герца, взяв в качестве веса вершины, получаемой в результате стягивания бикомпоненты, сумму весов входящих в нее вершин. Однако такой подход не охватывает все возможные комбинации контуров, которые могут встретиться в бикомпонентах.

Пусть теперь требуется найти $s-t$ -путь с наибольшим весом, содержащий как можно меньше вершин из $V \setminus C$. Чтобы удовлетворить этому ограничению, припишем каждой вершине, не принадлежащей C , небольшой отрицательный вес $(-\varepsilon)$, причем подберем ε так, чтобы выполнялось неравенство $1/\varepsilon > \min_{v_c \in C} \{w(v_c)\} \cdot |V|$.

Тем самым задача снова свелась к задаче о критическом пути.

3.2. Минимальное покрытие вершин орграфа путями. Рассмотрим задачу отыскания такого минимального по числу путей покрытия, а именно подмножества множества всех $s-t$ -путей, что каждая вершина орграфа принадлежит хотя бы одному такому пути.

Путевое покрытие (или покрытие путями) вершин графа $G = (V, E)$ есть множество путей $P = \{p_1, \dots, p_k\}$, таких, что для каждой вершины $v_i \in V$ существует путь $p_j \in P$, которому эта вершина принадлежит. Покрытие P называется *минимальным*, если не существует покрытия P' такого, что $|P'| < |P|$, где $|P|$ есть мощность множества P .

Множество S вершин графа G называется *несравнимым*, если для любой пары вершин v_i и v_j из S ни v_i не достижима из v_j , ни v_j не достижима из v_i . Вершины из множества S называются *несравнимыми*.

Пусть $i_v(G)$ — мощность наибольшего несравнимого множества вершин в G , и пусть $p_v(G)$ — наименьшее число путей, необходимых для покрытия вершин G .

Теорема 3. *Минимальное число непересекающихся путей, которые в совокупности покрывают все вершины бесконтурного орграфа G , равно максимально возможному числу вершин в подмножестве попарно несравнимых вершин, если это число конечно.*

Другими словами, $i_v(G) = p_v(G)$ для любого бесконтурного графа.

Обобщением теоремы 3 на случай произвольных орграфов служит

Теорема 4. *Для любого орграфа $i_v(G) = p_v(G)$.*

Доказательство проведем индукцией по числу бикомпонент. В орграфе $G = (V, E)$ обозначим через $\langle V' \rangle$ подграф, порожденный множеством вершин V' , $V' \subset V$. Пусть теперь подмножества V_1, \dots, V_k порождают бикомпоненты в G .

Базис ($k=1$). Граф G сильно связан. Легко показать, что существует путь $p[v_i, v_j]$, который покрывает все вершины в G для любых v_i и v_j . Таким образом, $p_v(G) = 1$. Так как любые две вершины в бикомпоненте взаимно достижимы, то $i_v(G) = 1$, т. е. $i_v(G) = p_v(G)$.

Индуктивный шаг. Предположим, что $i_v(G) = p_v(G)$ для всех $k < n$. Пусть $k = n$. Рассмотрим орграф $G' = (V', E')$, где $V' = \{V'_1, \dots, V'_k\}$ и $(V'_i, V'_j) \in E'$, если вершина бикомпоненты $\langle V'_j \rangle$ достижима из некоторой вершины $\langle V'_i \rangle$ (граф Герца для G). Ясно, что G' — бесконтурный граф. Пусть S — несравнимое множество в G' , мощность которого равна $i_v(G')$, $|S| = i_v(G')$. Пусть $\langle D_n(S) \rangle$ — подграф графа G' , порожденный множеством вершин $D_n(S) = \{v_i | v_i \text{ достигает } S \text{ или } v_i \in S\}$, а $\langle D_d(S) \rangle$ — подграф графа G' , порож-

денный множеством вершин $D_d(S) = \{v_j \mid v_j \text{ достижима из } S \text{ или } v_j \in S\}$. Имеем два случая.

1. Предположим, что существует несравнимое множество S с $|S| = i_v(G')$, такое, что $|D_n(S)| < k$ и $|D_d(S)| < k$. Так как S есть подмножество множеств $D_n(S)$ и $D_d(S)$ и каждое несравнимое множество вершин в $\langle D_n(S) \rangle$ или $\langle D_d(S) \rangle$ есть несравнимое множество вершин в G' , то $i_v(\langle D_n(S) \rangle) = i_v(\langle D_d(S) \rangle) = i_v(G') = |S|$. По индуктивному предположению существуют покрытия P_n и P_d вершин из $\langle D_n(S) \rangle$ и $\langle D_d(S) \rangle$ соответственно, такие, что $|P_n| = |P_d| = |S|$ и $p_v(\langle D_n(S) \rangle) = i_v(\langle D_n(S) \rangle) = i_v(\langle D_d(S) \rangle) = p_v(\langle D_d(S) \rangle) = i_v(G')$. Пути из P_n заканчиваются в вершинах из S , а пути из P_d начинаются в вершинах из S . Мы можем тогда соединить эти пути и образовать покрытие вершин из G' , для которого $p_v(G') = i_v(G')$. Но $i_v(G') = i_v(G)$, потому что если $\{V', \dots, V_i\}$ — несравнимое множество в G' , то, выбрав по вершине из каждого множества V_1, \dots, V_i , мы получим несравнимое множество для G ; обратно, если $\{v_1, \dots, v_j\}$ — несравнимое множество вершин в G и $v_1 \in V_1, \dots, v_j \in V_j$, то $\{V_1, \dots, V_j\}$ — несравнимое множество для G' . Более того, если $p[V'_s, V'_t] = [V'_s(V'_s, V'_{i_1})V'_{i_1} \dots V'_{i_j}(V'_{i_j}, V'_t)V'_t]$ есть путь в G' , то существует путь в G , который покрывает все вершины во всех бикомпонентах $\langle V_s \rangle, \langle V_{i_1} \rangle, \dots, \langle V_{i_j} \rangle, \langle V_t \rangle$ в G . Это непосредственно следует из того факта, что в бикомпоненте $\langle V_j \rangle$ существует путь $p[v_a, v_b]$, покрывающий все вершины $\langle V_j \rangle$ для любых v_a и v_b в V_j . Таким образом, для данного покрытия вершин из G' легко можно построить покрытие той же мощности для вершин из G . Так как $p_v(G)$ не может быть меньше, чем $i_v(G) = i_v(G')$, то отсюда следует, что $p_v(G) = i_v(G)$.

2. Предположим, что не существует несравнимого множества S с $|S| = i_v(G')$ такого, что $|D_n(S)| < k$ и $|D_d(S)| < k$. Тогда каждое максимальное несравнимое множество S должно быть или множеством входов в G' , или множеством выходов в G' . В любом случае пусть v'_s — вход, а v'_t — выход в G' и пусть $p[v'_s, v'_t]$ — путь в G' . Рассмотрим орграф $G'_1 = G' \setminus \{v'_s, v'_t\}$. Ясно, что число вершин в G'_1 равно $k-2$ и по индуктивному предположению существует покрытие вершин из G'_1 , для которого $i_v(G'_1) = p_v(G'_1)$. Можно легко убе-

даться в том, что $i_v(G'_1) = i_v(G') - 1$. Тогда, добавляя $p[v'_s, v'_t]$ к покрытию вершин из G'_1 , мы получим покрытие вершин из G' , для которого $i_v(G') = p_v(G')$. Как и в предыдущем случае, мы покажем, что $i_v(G') = i_v(G)$ и $p_v(G') = p_v(G)$.

Доказательство теоремы 4 не предлагает эффективного алгоритма для определения величин $i_v(G)$ и $p_v(G)$, но делает очевидным тот факт, что достаточно построить покрытие вершин для графа Герца, чтобы получить покрытие вершин для исходного графа.

3.3. Метод минимального потока построения покрытия. Преобразуем вначале бесконтурный орграф $G = (V, E)$ в потоковую сеть. Для этого каждую вершину $v_i \in V \setminus \{s, t\}$ расцепим на две вершины v'_i и v''_i и

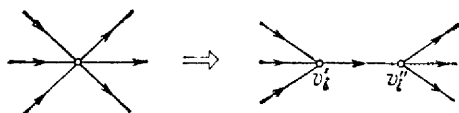


Рис. 3.7.

добавим дугу (v'_i, v''_i) , как показано на рис. 3.7. Получившийся граф обозначим через $G_f = (V_f, E_f)$. Сопоставим каждой дуге (a, b) графа G_f величину r_{ab} следующим образом:

$$r_{ab} = \begin{cases} 0, & \text{если } a = s \text{ или } b = t; \\ 0, & \text{если } (a, b) = (v''_i, v'_j), v''_i, v'_j \in V_f; \\ 1, & \text{если } (a, b) = (v'_i, v''_i), v'_i, v''_i \in V_f. \end{cases}$$

Полученный взвешенный граф будет представлять собой потоковую сеть, если величины r_{ab} мы будем интерпретировать как ограничение на величину потока по дуге.

Потоком по сети с ограничениями r_{ab} на поток снизу называется целочисленная функция $f(e)$, определенная на множестве дуг сети E и удовлетворяющая условиям

$$f(a, b) \geq r_{ab}, \quad (a, b) \in E_f; \quad (3.2)$$

$$\sum_{v_i \in A(v_j)} f(v_j, v_i) = \sum_{v_i \in B(v_j)} f(v_i, v_j) \quad \forall v_j \in V_f \setminus \{s, t\}. \quad (3.3)$$

Величина $F = \sum_{v_i \in A(s)} f(s, v_i) = \sum_{v_i \in B(t)} f(v_i, t)$ называется *величиной потока*. Поток f по сети с ограничениями на поток снизу называется *минимальным потоком*, если не существует потока f' с меньшей величиной, который бы удовлетворял условиям (3.2) и (3.3).

Теорема 5. Пусть f — минимальный поток в G_f ; тогда $F = p_v(G) = i_v(G)$.

Доказательство. Так как все ограничения на поток — целые числа и в G_f отсутствуют контуры, мы можем сопоставить каждой единице потока единственный $s-t$ -путь. Множество этих путей образует покрытие вершин в G_f . Ясно, что $F \geq i_v(G_f)$, так как никакие две вершины в несравнимом множестве не могут принадлежать одному и тому же пути. Если мы установим единичный поток на каждом пути p_i в минимальном покрытии вершин в G_f , мы получим допустимый поток. Таким образом, $F \leq p_v(G_f)$, а отсюда следует, что $F = p_v(G_f) = i_v(G_f)$, а из конструкции G_f следует, что $p_v(G) = p_v(G_f)$ и $i_v(G) = i_v(G_f)$.

3.4. Метод максимального паросочетания для отыскания покрытия. Пусть $G = (V, E)$ — бесконтурный орграф, и пусть $|V| = n$. Построим двудольный граф $G' = (V', E')$, в котором $V' = X \cup Y = \{x_1, \dots, x_n\} \cup \{y_1, \dots, y_n\}$ и $(x_i, y_j) \in E'$ тогда и только тогда, когда y_j достижима из x_i в G . Паросочетанием в G' называется подмножество дуг графа G' , не имеющих общих вершин. Паросочетание с наибольшим числом дуг называется *максимальным паросочетанием*.

Теорема 6. Пусть $G = (V, E)$ — бесконтурный орграф, $G' = (X \cup Y, E')$ — соответствующий двудольный граф, и пусть M — максимальное паросочетание в G' . Тогда $i_v(G) = |V| - |M|$.

Доказательство. Построим орграф $G'' = (V, E'')$, где $(v_i, v_j) \in E''$, если $(x_i, y_j) \in M$. Из определения паросочетания следует, что G'' состоит из g компонент, каждая из которых есть простой путь или изолированная вершина. Мы утверждаем, что $g = |V| - |M|$. Существуют в точности $|V| - |M|$ вершин в X , которые не являются концевыми вершинами дуг из M . Если x_i — такая вершина, а y_i — конец некоторой дуги в M , то отсюда следует, что v_i есть последняя вершина простого пути в G'' , и обратно. Если обе эти вершины x_i и y_i не являются концами дуг в M ,

то v_i — изолированная вершина в G'' , и обратно. Таким образом, $g = |V| - |M|$.

Будем считать изолированные вершины в G'' путями нулевой длины. Тогда, поскольку для каждого пути $p'' = [v_i(v_i, v_j)v_j \dots v_{k-1}(v_{k-1}, v_k)v_k]$ в G'' существует путь $p = [p[v_i, v_j]p[v_j, v_{k-1}]p[v_{k-1}, v_k]]$ в G , мы можем легко построить покрытие вершин из G , такое, что $|P| = |V| - |M|$. Чтобы показать, что P есть минимальное покрытие, предположим, что существует покрытие P' с $|P'| < |P|$. Рассмотрим оргграф G_1 , получаемый из G добавлением к E всех дуг (v_i, v_j) , если v_j достижима из v_i , т. е. транзитивное замыкание. Для данного P' можно легко построить покрытие P'_d различными путями вершин из G_1 , мощность которого $|P'_d|$ равна $|P'|$. В этом случае существует паросочетание M' в G' , такое, что каждая дуга в M' соответствует дуге в пути из P'_d и наоборот. Имеем, что $|M'| = \sum_{p \in P'_d} l(p) = |V| - |P'_d|$, где $l(p)$ — длина пути

p в P'_d . Но $|P'_d| = |P'| < |P|$ и $|M'| = |V| - |P'_d| > |M|$, что противоречит предположению о максимальной паросочетанности M в G' .

Трудоемкость метода зависит от трудоемкости алгоритма для решения задачи о максимальном паросочетании. Наилучший известный алгоритм имеет трудоемкость $O(n^{1/2}m)$ или $O(n^{5/2})$, так как в нашем случае $m = O(n^2)$.

3.5. Задача о требуемых путях. Пусть $G = (V, E)$ — оргграф, и пусть $R = \{\pi_1, \dots, \pi_k\}$ — множество некоторых путей в G . Будем говорить, что $s-t$ -путь p_i накрывает путь $\pi_i \in R$, если p_i представим в виде $[\alpha\beta\gamma]$, где α и γ — простые пути. Задача о требуемых путях состоит в следующем: найти минимальное по мощности множество $s-t$ -путей $P = \{p_1, \dots, p_m\}$, такое, что для любого $\pi_i \in R$ найдется путь $p_j \in P$, накрывающий его.

Мы можем взять вместо G его граф Герца G^* и заменить требуемые пути в G эквивалентными путями в G^* . Будем считать, что ни один требуемый путь π_i не является подпутем π_j . Пусть $\pi_i = [v_{i_1}(v_{i_1}, \dots, v_{i_t})v_{i_t}]$ и $\pi_j = [v_{j_1}(v_{j_1}, \dots, v_{j_t})v_{j_t}]$ — требуемые пути. Будем говорить, что π_j достижим из π_i , если:

- а) v_{j_1} достижима из v_{i_t} или $v_{j_1} = v_{i_t}$

либо

б) $\pi_i = \pi_{i_1} \pi_{i_2}$, $\pi_j = \pi_{j_1} \pi_{j_2}$ и $\pi_{i_2} = \pi_{j_1}$.

Прямым следствием этого определения является следующая лемма.

Лемма 7. *Отношение достижимости для требуемых путей определяет частичный порядок на множестве требуемых путей.*

Чтобы решить задачу о требуемых путях, построим оргграф $G_R = (V_R, E_R)$, где $V_R = \{v_1, \dots, v_k\}$ и $(v_i, v_j) \in E_R$, если π_j достижим из π_i в G^* . Таким образом, требуемые пути в G^* соответствуют вершинам в G_R . Оргграф G_R не содержит контуров, и для каждого пути в G_R найдется соответствующий путь в G .

Теорема 7. *Пусть P_R — минимальное покрытие вершин из G_R . Тогда найдется соответствующее минимальное покрытие P для требуемых путей в G^* , такое, что $|P| = |P_R|$.*

Следовательно, задача о требуемых путях оказалась сведенной к задаче нахождения минимального покрытия вершин.

3.6. Минимальное покрытие дуг оргграфа путями. Задача состоит в отыскании минимального по числу путей подмножества всех путей, такого, что каждая дуга принадлежит хотя бы одному такому пути. При этом возможно дополнительное требование о том, чтобы все пути исходили из одной вершины. Рассмотрим вначале покрытие дуг оргграфа путями, исходящими из входа оргграфа. Пусть P — некоторое покрытие дуг оргграфа. Назовем число путей в P *кратностью* покрытия P , а сумму длин всех путей из P — *длиной* покрытия P . Ясно, что покрытие P оргграфа G существует, если каждая его вершина, из которой исходит хотя бы одна дуга, достижима из начальной вершины. Можно показать, что это условие и достаточно. Будем называть граф, для которого существует покрытие, *правильным*.

Покрытие $P = \{p_1, \dots, p_k\}$ оргграфа G называется *приведенным*, если выполняются следующие условия:

а) для всех $i \neq j$, $1 \leq i, j \leq k$, путь p_i не является начальным отрезком пути p_j ;

б) для всякого q , являющегося начальным отрезком пути p_i , $q \neq p_i$, $i \in [1:k]$, множество $P' = \{p_1, \dots, p_{i-1}, q, p_{i+1}, \dots, p_k\}$ не является покрытием.

Пусть P — приведенное покрытие оргграфа G . Из условия б) следует, что если $p \in P$ и заканчивается

дугой u , то эта дуга не принадлежит никакому другому пути из P . Отсюда непосредственно вытекает, что для правильного графа всегда существует приведенное покрытие, кратность которого не превосходит числа дуг орграфа. Из определения приведенного покрытия следует также, что минимальное по длине покрытие является приведенным.

Приведенное покрытие орграфа, являющееся минимальным по кратности и по длине, называется *минимальным*.

Можно показать, что минимальное покрытие орграфа существует не всегда. В самом деле, для графа

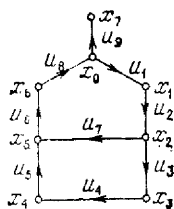


Рис. 3.8.

на рис. 3.8 существуют два приведенных покрытия: $P_1 = \{[u_1, u_2, u_3, u_4, u_5, u_6, u_8, u_1, u_2, u_7, u_8, u_8, u_9]\}$ и $P_2 = \{[u_1, u_2, u_3, u_4, u_5, u_6, u_8, u_9], [u_1, u_2, u_7]\}$. Покрытие P_1 минимально по кратности (кратность его равна 1), но не минимально по длине (длина его равна 13). Покрытие P_2 является минимальным по длине (длина 11), но не минимальным по кратности (крат-

ность 2). Легко установить, что для данного орграфа минимального покрытия не существует.

Сформулируем достаточные условия существования минимального покрытия. Пусть $\Delta(v) = \deg^+(v) - \deg^-(v)$. Назовем вершину v *положительной*, если $\Delta(v) > 0$, и *отрицательной* в противном случае. Множество всех положительных вершин обозначим через V^+ , множество отрицательных — через V^- . Обозначим теперь через R^+ подмножество вершин из множества V^+ , из которых не достижима ни одна отрицательная вершина.

Теорема 8. *Орграф G имеет минимальное покрытие, если для него $R^+ = V^+$.*

Доказательство. Если у правильного графа G множество V^+ пусто, т. е. для всех вершин полустепень захода равна полустепени исхода, то по теореме 18 главы 2 G обладает эйлеровым контуром. Очевидно, что этот эйлеров контур является минимальным покрытием.

Предположим теперь, что V^+ не пусто и, следовательно, V^- также не пусто. Поскольку $V^+ = R^+$, а множество R^+ состоит из вершин, из которых не достижима ни одна отрицательная вершина, то граф G заведомо

не сильно связан. Покажем, что кратность такого покрытия равна $\pi(G) = \sum_{v \in V^+} \Delta(v)$.

Вначале покажем, что кратность покрытия не превосходит $\pi(G)$. Для этого проведем из каждой вершины $v \in V^+$ в вершину v_0 (рассматриваемую как вход) ровно $\Delta(v)$ дуг вида (v, v_0) . В полученном сильно связном мультиграфе H зафиксируем контур, проходящий через все дуги H . Этот контур не содержит двух следующих друг за другом дополнительных дуг, более того, его можно выбрать так, чтобы каждая дополнительная дуга входила в него ровно один раз. Наличие такого контура гарантирует существование покрытия, кратность которого не превосходит $\pi(G)$.

Пусть теперь $P = \{p_1, \dots, p_n\}$ — некоторое покрытие графа G . Замкнем каждый путь p_i , добавив дугу (v_i, v_0) из вершины v_i , в которой этот путь заканчивается. Обозначим через C контур в полученном графе H , проходящий через все дуги H и соответствующий покрытию P . Превратим H в мультиграф H' , заменив каждую дугу u орграфа G столькоими копиями, во сколько путей из покрытия P она входит, или, другими словами, сколько раз она встречается в контуре C . В силу теоремы 18 главы 2 мультиграф H' обладает эйлеровым контуром. Поскольку для любой вершины v мультиграфа H' имеем $\Delta(v) = 0$, то в вершину v_0 заходит не менее $\sum_{v \in R^+} \Delta(v)$ замыкающих дуг, откуда и следует, что кратность покрытия P не менее $\sum_{v \in R^+} \Delta(v)$.

Так как $R^+ = V^+$, то отсюда вытекает оценка кратности минимального покрытия.

Пусть теперь P есть покрытие орграфа G минимальной длины d , и пусть каждый путь из P заканчивается в вершине, отличной от v_0 . Более того, он заканчивается в вершине из R^+ , так как в противном случае длину покрытия можно уменьшить. Предположим, что P не минимально по кратности, т. е. его кратность превышает $\pi(G)$. Построим сильно связный граф H , замкнув каждый путь дугой, ведущей в v_0 , и построим для него эйлеров мультиграф H' . Поскольку кратность покрытия P больше $\pi(G)$, то в G найдется вершина $\bar{v} \in V^+$, из которой в графе H выходит l замыкающих дуг, причем $l > \Delta(\bar{v})$. Следовательно, в графе H найдется дуга $u \in G$, заходящая в вершину

\bar{t} и принадлежащая по крайней мере двум путям из P . Тогда из графа H' удалим одну дугу u' , соответствующую дуге u , и удалим одну замыкающую дугу, исходящую из \bar{t} . Затем из начала дуги u' в вершину v_0 проведем одну дугу. Легко увидеть, что полученный граф H'' является эйлеровым и эйлерову контуру графа H'' соответствует покрытие графа G длины $d-1$. Следовательно, P не является покрытием минимальной длины.

Рассмотрим, наконец, минимальное по длине покрытие $P = \{p_1, \dots, p_k\}$ графа G , у которого некоторые пути заканчиваются в v_0 . Пусть кратность покрытия P больше $\lambda(G)$ и пути p_1, \dots, p_i заканчиваются в v_0 . Легко увидеть, что покрытие $P' = \{[p_1 p_2 \dots p_{i+1}], p_{i+2}, \dots, p_k\}$ также является покрытием минимальной длины и все пути из P' заканчиваются только в положительных вершинах графа G . Теорема доказана, так как кратность такого покрытия P' минимальна.

Опишем алгоритм построения приведенного покрытия правильного графа G . Пусть пути искомого покрытия обозначаются p_i , а множество вершин графа G , из которых исходит хотя бы одна дуга, обозначается T . Каждой вершине $x \in T$ поставим в соответствие множество $\bar{A}(x)$ дуг, исходящих из x . Дуги в $\bar{A}(x)$ занумерованы числами $1, 2, \dots$, причём если из двух дуг u_i и u_j из $\bar{A}(x)$ одна, скажем u_i , является петлей, то $i < j$. Пусть, кроме того, $\varphi(u)$ обозначает номер дуги u в множестве $\bar{A}(x)$, а $\delta(p)$ — конечную вершину пути p , исходящего из вершины x . В алгоритме, кроме построения покрытия, производится вычисление параметра j , необходимого для оценки длины полученного покрытия.

Вход: Граф $G = (V, E)$, заданный упорядоченными списками дуг $\bar{A}(v)$, исходящих из вершин.

Выход: Покрытие $P = \{p_i\}$.

начало

1. процедура УДЛИНЕНИЕ(v);

2. **начало**

3. выбрать из $\bar{A}(v)$ дугу u с наименьшим номером;

4. $p_i := p_i u$;

5. $\bar{A}(v) := \bar{A}(v) \setminus \{u\}$;

6. если $\varphi(u) \neq 1$ то $j := j + 1$;

7. если $\bar{A}(\delta(p_i)) \neq \emptyset$ то УДЛИНЕНИЕ($\delta(p_i)$)

8. иначе если из $\delta(p_i)$ достижима хотя бы одна вершина y с $\bar{A}(y) \neq \emptyset$ то

9. начало

10. построить кратчайший путь q_j из $\delta(p_i)$ в y ;
 11. $p_i := p_i q_j$;
 12. УДЛИНЕНИЕ($\delta(p_i)$);
 13. конец
 14. иначе добавить p_i к P ;
 15. конец УДЛИНЕНИЕ;
 16. $P := \emptyset$; $i := 1$; $j := 0$;
 17. для всех $x \in T$ цикл $A(x) := \bar{A}(x)$;
 18. $x := s$;
 19. УДЛИНЕНИЕ(s);
 20. $i := i + 1$;
 21. если $\bigcup_{x \in T} A(x) = \emptyset$ то стоп
 22. иначе переход к шагу 18;
- конец

Теорема 9. Для правильного графа всегда существует покрытие длины, не превосходящей величины $m + \sum_{i=1}^j (i-1)(m_i-1)$, где m — число дуг орграфа, а m_i — мощность множества $\bar{A}(x_i)$, $x_i \in T$.

Доказательство. Упорядочим множество $\{m_i\}_{x_i \in T}$ отношением \leq и обозначим упорядоченное таким образом множество через $\{m_1, \dots, m_j\}$, причем $m_i \leq m_j$, если $i < j$. Будем считать, что j -й шаг алгоритма составляют построения, выполняемые алгоритмом при фиксированном параметре j . Путь q_j , строящийся на шаге 10 алгоритма, назовем транслирующим путем. Очевидно, что длина покрытия равна сумме числа дуг графа и длины всех транслирующих путей. Из описания алгоритма следует, что длина транслирующего пути q_j не превосходит числа таких вершин $v \in T$, для которых на j -м шаге $\bar{A}(v) = \emptyset$. Поэтому для значений j , $j \in [0; m_1 - 1]$, длина q_j равна нулю, а для значений j из интервала $[m_1; m_2 - 1]$ длина q_j не превосходит единицы. На последнем шаге работы алгоритма значение j равно $\sum_{i=1}^j (m_i - 1)$. Отсюда видно, что суммарная длина транслирующих путей не превышает величины $Q = \sum_{i=1}^j (i-1)(m_i-1)$.

В случае бесконтурного графа минимальное по кратности покрытие дуг орграфа можно построить с

помощью минимального потока по сети. Для этого достаточно приписать каждой дуге орграфа ограничение на поток снизу r_{ab} , равное единице. Для графов с контурами, как и в случае покрытий вершин, необходимо построить сначала минимальное по кратности покрытие дуг для графа Герца, а затем использовать тот факт, что все дуги бикомпоненты можно покрыть одним путем.

§ 4. Отыскание доминаторов в орграфе

Вершина $D(v)$ называется *доминатором* для вершины v , если $D(v)$ лежит на каждом пути из входа графа в вершину v . Вершина $Id(v)$ называется *непосредственным доминатором* для v , если любой другой доминатор для v является также доминатором для $Id(v)$.

В этом параграфе описывается алгоритм отыскания непосредственных доминаторов для каждой вершины графа. Трудоемкость алгоритма — $O(n \log n + m)$. Если $m \geq n \log n$, то алгоритм имеет трудоемкость $O(m)$ и оптимален с точностью до постоянной.

4.1. Преобразования, сохраняющие доминаторы. Предположим, что в графе $G = (V, E)$ с входом s выполнен поиск в глубину, в результате чего все вершины получили M - и N -номера, а все дуги оказались разбитыми на классы древесных, прямых, поперечных и обратных дуг. Для каждой вершины v определим тогда множество $F(v)$, где $F(v) = \{w \mid w \neq v \text{ и существует вершина } u, \text{ достижимая из } w \text{ по древесным дугам по пути, проходящему через } v, \text{ и такая, что } (u, w) \text{ есть обратная дуга в } G\}$. Пусть вершина $H(v)$ имеет наибольший M -номер в $F(v)$, если $F(v) \neq \emptyset$. (Здесь и далее мы будем использовать только M -номера. При необходимости мы будем идентифицировать вершины M -номерами.) Ясно, что вершина $H(v)$ достижима из w по T , если $w \in F(v)$. Рассмотрим преобразование: удалить из G все обратные дуги и добавить новые обратные дуги $(v, H(v))$ для каждой вершины v , для которой $H(v)$ определено. Это преобразование носит название *замены обратных дуг*.

Пример. После замены обратных дуг в графе на рис. 3.9 вершина K потеряла обратную дугу, а вершины I , F и C приобрели. Древесные дуги графа изображены на рисунке сплошными жирными линиями,

прямые — сплошными тонкими, поперечные — волнистыми, обратные — штриховыми. Первое число из пары чисел, сопоставленных каждой вершине, есть M -номер, второе — N -номер.

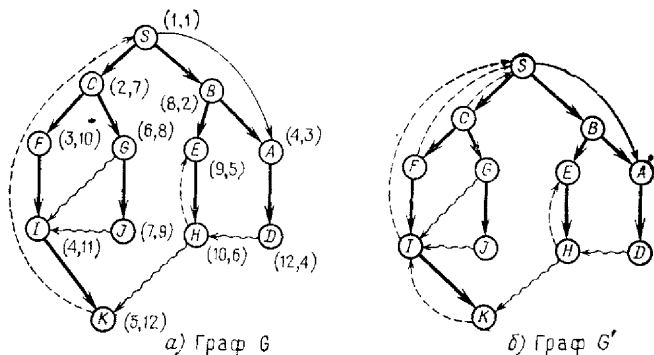


Рис. 3.9.

Лемма 8. Пусть G' — граф, получаемый из G заменой обратных дуг, и пусть T — дерево поиска в глубину для G . Если (u, w) — обратная дуга в G и в T существует путь из w в u , содержащий вершину v , то в G' существует путь из v в w , содержащий только обратные дуги.

Доказательство проводится индукцией по длине пути в G из w в v . Детали предоставляются читателю.

Лемма 9. Вершина w является доминатором вершины v в G тогда и только тогда, когда w доминирует v в G' .

Доказательство. Предположим, что w не доминирует v в G . Тогда в G существует путь из s в v , не содержащий w . Предположим, что этот путь содержит обратную дугу (u, u') , причем w достижима из u' по T . Тогда мы можем заменить часть пути, начинающуюся в s и включающую последнюю такую дугу, путем из древесных дуг. Это дает нам путь в G из s в v , который не содержит ни w , ни одной обратной дуги (u, u') , такой, что w достижима из u' по T . Если мы теперь заменим каждую оставшуюся обратную дугу в пути соответствующим путем из обратных дуг в G' , что гарантируется леммой 8, мы получим путь в G' из s в v , который не содержит w , откуда и следует, что w не доминирует v в G' .

Обратно, предположим, что w не доминирует v в G' . Тогда существует путь p в G' из s в v , не содержащий w . Предположим, что этот путь содержит обратную дугу $(u, H(u))$, причем w достижима из $H(u)$ по T . Тогда мы можем заменить часть p' этого пути, расположенную ближе к s и включающую последнюю такую дугу, путем из древесных дуг. Это дает нам путь p'' в G' из s в v , не содержащий ни w , ни одной обратной дуги $(u, H(u))$, такой, что w достижима из $H(u)$ по T . Для любой обратной дуги $(u, H(u))$ в пути p'' существует соответствующая ей обратная дуга $(u', H(u))$ в G , причем w достижима из $H(u)$ по T . Если мы заменим каждую обратную дугу $(u, H(u))$ в p'' путем из древесных дуг из u в u' и следующей за ним обратной дугой $(u, H(u))$, то мы получим путь p в G из s в v , не содержащий w . Отсюда следует, что w не доминирует v в G' . Лемма доказана.

Пусть теперь G есть граф, к которому уже применено преобразование замены обратных дуг. В соответствии с соглашением о возможной идентификации вершин их M -номерах будем писать $v < w$, понимая под этим, что $M(v) < M(w)$. Пусть (u, v) и (u_1, v) — две прямые дуги в G , причем $u_1 > u$. Рассмотрим преобразование: удалить прямую дугу (u_1, v) .

Лемма 10. Пусть G' — граф, получаемый из G удалением прямых дуг. Вершина w доминирует вершину v в G тогда и только тогда, когда w доминирует v в G' .

Доказательство. Так как G' есть суграф G , то каждый путь в G' есть путь в G . Таким образом, если w доминирует v в G , то w доминирует v в G' .

Обратно, предположим, что w не доминирует v в G . Тогда существует путь p в G из s в v , не содержащий w . Если p не содержит дугу (u_1, v) , то p есть путь в G' и w не доминирует v в G' . Предположим, что p содержит дугу (u_1, v) . Если w — потомок u_1 , то мы можем заменить часть пути p ближе к s , включающую в себя дугу (u_1, v) , путем из древесных дуг из s в u и следующей за ним прямой дугой (u, v) , получив в результате путь p' в G' из s в v , не содержащий w . Если w не является потомком u_1 , мы заменим (u_1, v) в p путем из древесных дуг из u_1 в v и получим путь p' в G' из s в v , не содержащий w ; в любом случае

w не может доминировать v в G' , что и доказывает лемму.

Пусть v — вершина в G , такая, что из нее выходит одна обратная дуга $(v, H(v))$, заходит самое большее одна прямая дуга и не заходят ни поперечная, ни обратная дуги. Преобразование «удаление обратной дуги» состоит в удалении обратной дуги $(v, H(v))$ и добавлении дуги $(u, H(v))$, если (u, v) существует и $(u, H(v))$ есть обратная дуга, т. е. если $u < H(v)$.

Лемма 11. Пусть G' — граф, получаемый из G удалением обратных дуг. Тогда вершина d доминирует w в G' тогда и только тогда, когда d доминирует w в G . При этом предполагается, что ни одно удаление обратной дуги не применялось к вершинам x с номерами меньшими, чем номер вершины v .

Доказательство. Предположим, что d не доминирует w в G . Тогда существует путь p в G из s в w , не содержащий d . Если p не содержит $(v, H(v))$, то p есть путь в G' . Если p содержит $(v, H(v))$, то p должен содержать либо дугу (u, v) , либо древесную дугу, заходящую в v . Если d не есть непосредственный предок для $H(v)$, то мы можем заменить часть p ближе к s , содержащую дугу $(v, H(v))$, путем из древесных дуг из s до $H(v)$; в результате получим путь в G' , не содержащий d . Предположим теперь, что d есть непосредственный предок для $H(v)$. Если дуга перед дугой $(v, H(v))$ в p есть (u, v) и $u < H(v)$, то мы можем заменить (u, v) и $(v, H(v))$ дугой $(u, H(v))$ и получить путь в G' , не содержащий d . Если дуга перед $(v, H(v))$ есть древесная дуга или дуга (u, v) и $u \geq H(v)$, то по лемме 8 мы можем заменить $(v, H(v))$ и дугу перед ней путем из обратных дуг и получить путь в G' , который не содержит d . В любом случае d не доминирует w в G' .

Обратно, предположим, что d не доминирует w в G' . Тогда существует путь p' в G' из s в w , не содержащий d . Если p' не содержит $(u, H(v))$, то p' есть путь в G . Предположим, что p' содержит $(u, H(v))$. Если $d \neq v$, то мы можем заменить $(u, H(v))$ в p' на (u, v) и $(v, H(v))$, что даст нам путь в G , не содержащий d . Если $d = v$, то мы можем заменить часть пути p' ближе к s , содержащую дугу $(u, H(v))$, путем из древесных дуг из s в $H(v)$, что даст нам путь в G , не содержащий d . В любом случае d не доминирует w в G , что и доказывает лемму.

Пусть $G(i)$ есть суграф графа G , порожденный древесными дугами и дугами, заходящими в вершины v , где $M(v) > i$. Тогда i -м семидоминатором $Sd(i, v)$ вершины v называется ее непосредственный доминатор $Id(v)$ в $G(i)$.

В дальнейшем будем предполагать, что $v \neq 1$, так что все семидоминаторы определены.

Лемма 12. Если $v \neq 1$ и $i \geq v$, то $Sd(i, v)$ есть непосредственный предок вершины v в дереве T поиска в глубину.

Лемма 13. Если $v \neq 1$, то для всех i вершина $Sd(i, v)$ достижима из $Sd(i-1, v)$ по T .

Справедливость данной леммы вытекает из того, что $G(i)$ есть подграф графа $G(i-1)$.

Лемма 14. Если $v \neq 1$ и $Sd(i, v) = i$, то $Id(v) = 1$.

Доказательство. Из леммы 13 следует, что $Sd(i, v)$ достижима из $Sd(0, v)$ по T и $Sd(0, v) = Id(v)$. Таким образом, если $Sd(i, v)$ доминирует v в G , то $Sd(i, v) = Id(v)$. Теперь покажем, что $Sd(i, v)$ доминирует v в G , если $Sd(i, v) = i$.

Предположим, что существует путь p в G из s в v , не содержащий $i = Sd(i, v)$. Тогда v должна быть достижима из i по T . При этом некоторая дуга пути p должна начинаться в вершине, не являющейся потомком вершины i , и вести в вершину — потомок i . Пусть (u, w) — последняя такая дуга в p . Тогда $w > i$ и все дуги, следующие за (u, w) в p , имеют оба конца среди потомков i . Так как все потомки вершины i имеют M -номера, большие i , и так как u не является потомком i , то мы можем заменить часть p от начала пути до дуги (u, w) путем из древесных дуг из s в u и получить путь в $G(i)$, не содержащий i . Но это противоречит определению $Sd(i, v)$, поэтому вершина $Sd(i, v)$ должна доминировать v в G . Лемма доказана.

Лемма 15. Если $v \neq 1$ и $i \leq v$, то либо $Sd(i, v) = Id(v)$, либо i достижима из $Sd(i, v)$ по T .

Доказательство проводится индукцией по i с использованием лемм 12—14.

Лемма 16. Если u не доминирует v в G и дуга (u, v) в G заменена дугой $(Id(u), v)$, то d доминирует w в G тогда и только тогда, когда d доминирует w в результирующем графе G' .

Доказательство. Предположим, что d не доминирует w в G . Тогда существует путь p в G из s в w , не содержащий d . Если p не содержит дугу (u, v) ,

то p — путь в G . Предположим, что p содержит дугу (u, v) . Тогда p содержит $Id(u)$ и мы можем заменить часть пути p от $Id(u)$ до v дугой $(Id(u), v)$ и получить путь в G' из s в w , не содержащий d . Таким образом, d не доминирует w в G' .

Обратно, предположим, что d не доминирует w в G' . Тогда существует путь p' в G' из s в w , не содержащий d . Если $(Id(u), v)$ не лежит на пути p' , то p' есть путь в G . Предположим, что $(Id(u), v)$ лежит на пути p' . Если $d = u$, тогда как u не доминирует v , то существует путь q в G из s в w , не содержащий d . Заменяя путем q часть пути p' ближе к s , включающую дугу $(Id(u), v)$, получим путь p в G из s в w , не содержащий d . Предположим, что $d \neq u$. Если каждый путь из $Id(u)$ в u в графе G содержит d , то d доминирует u . Также и каждый путь из s в d должен содержать $Id(u)$, так как $Id(u)$ доминирует u . Тогда $Id(u)$ доминирует d , или $Id(u) = d$. И в том, и в другом случае приходим к противоречию. Отсюда следует, что существует некоторый путь r из $Id(u)$ в u в графе G , не содержащий d . Заменяя r и дугу (u, v) на дугу $(Id(u), v)$ в p' , получим путь в G из s в w , не содержащий d . Ни в одном случае d не может доминировать w в G . Лемма доказана.

Пусть (u, v) — поперечная дуга в графе G . Преобразование «замена поперечной дуги» состоит в удалении дуги (u, v) и добавлении дуги $(Sd(v, u), v)$.

Лемма 17. Пусть G' — граф, полученный из G заменой поперечной дуги. Тогда d доминирует w в G тогда и только тогда, когда d доминирует w в G' .

Доказательство. Если (u, v) — поперечная дуга, то $v < u$. Если $Sd(v, u) = Id(u)$, то справедливость леммы вытекает из леммы 16, так как u не может доминировать v , если только не имеет место достижимость v из u по T , что невозможно по определению поперечной дуги. Предположим, что $Sd(v, u) \neq Id(u)$. Тогда v достижима из $Sd(v, u)$ по древесным дугам по лемме 15. Теперь предположим, что d доминирует w в G . Тогда существует путь p в G из s в w , не содержащий d . Если (u, v) не является дугой в p , то p есть путь в G' . Пусть p содержит (u, v) . Если d не является предком $Sd(u, v)$, то мы можем заменить часть пути p ближе к s , включающую дугу (u, v) , путем из древесных дуг из s в $Sd(v, u)$ и

дугой ($Sd(v, u), v$) и получить путь в G' из s в w , не содержащий d .

С другой стороны, пусть d — предок $Sd(v, u)$. Рассмотрим часть пути p из s в u . Пусть (y, z) — последняя дуга в этой части пути p с $y \leq v$. Тогда вершина y должна быть предком вершины u , так как по свойствам поиска в глубину и M -нумерации любой путь из y в u должен проходить через общего предка a вершин y и u и эта вершина удовлетворяет соотношению $a \leq y \leq v$. Вершина y должна быть предком v , так как $y \leq v \leq u$, и любой предок вершины u , который не является предком вершины v , будет иметь номер больший, чем номер вершины v . Часть пути p от y до u лежит в подграфе $G(v)$, так что y достижима из $Sd(v, u)$ по древесным дугам. В противном случае будет существовать путь в $G(v)$ из s в u , не содержащий $Sd(v, u)$, что невозможно. Таким образом, на пути p найдется такая вершина y , что в дереве T существует путь $\mu[d, v] = (d, \dots, Sd(v, u), \dots, y, \dots, v)$. Мы можем заменить часть пути p от y до v , включающую дугу (u, v) , путем из древесных дуг из y в v . Это дает нам путь в G' из s в w , не содержащий d , и d не доминирует w в G' .

Обратно, предположим, что d не доминирует w в G' . Пусть p' — путь из s в w в графе G' , не содержащий d . Если p' не содержит дуги $(Sd(v, u), v)$, то p' есть путь в G . Предположим, что p' содержит $(Sd(v, u), v)$. Если $d = u$, то мы можем заменить дугу $(Sd(v, u), v)$ в p' путем из древесных дуг из $Sd(v, u)$ в v и получить путь из s в w в G , не содержащий d . Если $d \neq u$, то должен быть путь q в $G(v)$ (и, следовательно, в G) из вершины $Sd(v, u)$ в u , не содержащий d . Иначе $Sd(v, u)$ доминирует d и d доминирует u , что невозможно. Заменяя дугу $(Sd(v, u), v)$ в пути p' путем q и дугой (u, v) , получим путь из s в w в G , не содержащий d . В любом случае d не может доминировать вершину w в G , что и доказывает лемму.

4.2. Закончим теоретическое обоснование алгоритма вычисления доминаторов.

Лемма 18. Пусть i — вершина в G , такая, что в нее заходят самое большое две дуги — древесная дуга (u, i) и, возможно, прямая дуга (w, i) . Если ни одна прямая дуга не заходит в i , то $Sd(i-1, v) = Sd(i, v)$ для всех v . Если в i входит прямая дуга (w, i) , то $Sd(i-1, v) = w$ для всех v , которые дости-

жимы из i по T , и в T найдется путь из u в w , содержащий $Sd(i, v)$, и $Sd(i-1, v) = Sd(i, v)$ для всех остальных вершин.

Доказательство. Если в i не заходит ни обратная дуга, ни поперечная, ни прямая, то $G(i) = G(i-1)$; таким образом, первая часть леммы справедлива. Предположим, что в i заходит прямая дуга (w, i) . Так как $G(i-1)$ не имеет дуг, кроме древесных и дуг, которые ведут в вершины с номерами большими, чем номер i , то любой путь в $G(i-1)$, который содержит дугу (w, i) , должен заканчиваться в вершине — потомке i . Таким образом, пути из s в вершины, не являющиеся потомками i , одни и те же и в $G(i)$, и в $G(i-1)$. Это означает, что если v не является потомком i , то $Sd(i, v) = Sd(i-1, v)$.

Предположим, что v — потомок i . Если $Sd(i, v) \neq Sd(i-1, v)$, то должен быть путь в $G(i-1)$ из s в v , не содержащий $Sd(i, v)$, который, однако, не является путем в $G(i)$ с тем же свойством. Путь p должен содержать прямую дугу (w, i) . Но если в T не существует пути из w в u , содержащего $Sd(i, v)$, то мы можем заменить дугу (w, i) в p путем из древесных дуг из w в i и получить путь в $G(i)$ из s в v , который не содержит $Sd(i, v)$. Из этого противоречия следует, что если $Sd(i-1, v) \neq Sd(i, v)$, то существует путь в T из w в u , содержащий $Sd(i, v)$.

Предположим теперь, что такой путь в T существует. Если u достижима из $Sd(i, v)$ в дереве T , то вершина d доминирует $Sd(i, v)$ в подграфе $G(i)$ только в том случае, если $Sd(i, v)$ достижима из d по T и d доминирует v в $G(i)$. С другой стороны, любая вершина, которая не является предком $Sd(i, v)$, не может доминировать v в $G(i)$. Таким образом, в $G(i)$ доминаторы вершины v суть предки $Sd(i, v)$. Теперь в $G(i-1)$ вершина p доминирует вершины w и v , если w достижима из d по T . Если $Sd(i, v)$ достижима из d по T , но d не есть предок вершины w , то d не доминирует v , так как путь из древесных дуг из s в w с последующей прямой дугой (w, i) и путем из древесных дуг из i в v есть путь в подграфе $G(i-1)$, не содержащий d . Тогда доминаторы v в подграфе $G(i-1)$ суть предки w и $Sd(i-1, v) = w$.

Лемма 19. Пусть G' — граф, получаемый из графа G применением преобразования «удаление прямых дуг», а также преобразований «удаление обратных

дуг» и «замена поперечных дуг». Тогда множества семидоминаторов в G' и G совпадают.

Доказательство. Мы должны сравнить доминаторы в подграфах $G(i)$ и $G'(i)$, чтобы проверить, что i -е семидоминаторы в G и G' одни и те же. Пусть G' образуется из G удалением прямых дуг, и пусть (u_1, v) — удаленная прямая дуга, (u, v) — другая прямая дуга в G и $u_1 > u$. Если $i \geq v$, то $G(i) = G'(i)$, и лемма справедлива. Если $i < v$, то и (u, v) , и (u_1, v) имеются в $G(i)$, $G'(i)$ образуется из $G(i)$ удалением прямой дуги, утверждение леммы следует из леммы 10.

Предположим, что G' образуется из G удалением обратной дуги. Пусть $(v, H(v))$ — удаленная обратная дуга, а $(u, H(v))$ — добавленная обратная дуга, причем в v заходят только древесная дуга и прямая дуга (u, v) и выходит только одна обратная дуга. Если $i \geq H(v)$, то $G(i) = G'(i)$, и лемма справедлива. Если $i < H(v)$, то $(v, H(v))$ и (u, v) принадлежат $G(i)$, дуга $(u, H(v))$ принадлежит $G'(i)$ и $G'(i)$ образуется из $G(i)$ удалением обратной дуги. В этом случае лемма следует из леммы 11.

Предположим, что граф G' образован из G заменой поперечной дуги. Пусть (u, v) — поперечная дуга в G , которая заменена на дугу $(Sd(u, v), v)$, в результате чего получен G' . Если $i \geq v$, то $G'(i) = G(i)$, и лемма справедлива. Если $i < v$, то $G(v)$ есть суграф графа $G(i)$ и v -й семидоминатор вершины u один и тот же и в $G(i)$, и в G . Дуга (u, v) содержится в $G(i)$, и дуга $(Sd(v, u), v)$ — в $G'(i)$. Отсюда следует, что $G'(i)$ образуется из $G(i)$ заменой поперечной дуги, и лемма справедлива по лемме 17.

4.3. Алгоритм отыскания доминаторов. Приведем набросок алгоритма в алголоподобном виде.

процедура ДОМИНАТОРЫ(G, s);

начало

Шаг 1

1. Применим процедуру ПОИСК(v, n) для классификации дуг и построения M -нумерации вершин, достижимых из s ; пусть G_1 — подграф графа G , содержащий все вершины, достижимые из s ; пусть G_1 имеет n_1 вершин;

2. для каждой вершины v из G , но не из G_1 цикл $Id(v) := 0$;

Шаг 2

3. для каждой вершины v из G_1 цикл

4. вычислить $H(v)$;
5. применить замену обратных дуг к G_1 для получения графа G_2 ;

Шаг 3

6. для i от n_1 шаг -1 пока 0 цикл
7. для v от 1 до n_1 цикл
8. вычислить $Sd(i, v)$;
9. для i от 1 до n_1 цикл
10. $Id(i) := Sd(0, i)$;

конец

Полный текст алгоритма включает в себя процедуру поиска в глубину, которая сортирует обратные дуги по значению их второй вершины, а также используется для начала процесса удаления прямых дуг и для выработки информации о предках вершин для инициализации процесса вычисления семидоминаторов. Операции над множествами и над очередями здесь не реализуются, но предполагается выполнение примитивных операций. Однако время, требуемое этими операциями, включается в оценку времени всего алгоритма.

Вход: Граф G с входом s , заданный списками смежности $A(v)$.

Выход: Множество доминаторов.

процедура ДОМИНАТОРЫ(G, s);

начало

1. **процедура ПОИСК(v);**
2. **начало**

примечание здесь используется модифицированная процедура поиска в глубину; она нумерует вершины графа M -номераами, классифицирует дуги, удаляет все, кроме одной, прямые дуги $(\mathcal{L}(v), v)$, входящие в v , сортирует обратные дуги, вычисляет число потомков и предка каждой вершины в дереве поиска в глубину; вершины, не достигаемые при таком поиске, не имеют доминаторов; переменная \bar{m} обозначает последний M -номер \mathcal{M} , переменная \bar{n} обозначает последний N -номер \mathcal{N} ; $ND(v)$ обозначает число потомков вершины v в дереве T **конец примечания**

3. $\bar{m} := \mathcal{M}(v) := \bar{m} + 1$;
4. $ND(v) := 1$;
5. для $w \in A(v)$ цикл
6. **начало**
7. **если** $\mathcal{M}(w) = 0$ **то**
8. **начало**

9. $\text{ПРЕДОК}(w) := v$;
10. $\text{ПОИСК}(w)$;
11. $ND(v) := ND(v) + ND(w)$;
12. **конец**
13. **иначе если** $\mathcal{N}(w) = 0$ **то**
14. **начало**

примечание вершина w помещена в стек, а (v, w) есть обратная дуга **конец** примечания

15. добавить (v, w) к $\text{ГРУППА}(w)$;
16. **конец**
17. **иначе если** $\mathcal{M}(v) < \mathcal{M}(w)$ **то**
18. пометить (v, w) как прямую дугу;
19. **если** $\mathcal{M}(v) < \mathcal{L}(w)$ **то**
20. $\mathcal{L}(w) := \mathcal{M}(v)$;
21. **иначе** добавить (v, w) к списку поперечных дуг, входящих в w ;
22. $\bar{n} := \mathcal{N}(v) := \bar{n} - 1$;
23. **конец** цикла;
24. **конец** ПОИСК ;

Шаг 1

25. $\bar{m} := 0$; $\bar{n} := n + 1$;
26. **для** каждой вершины v **цикл**
27. **начало**
28. $\mathcal{M}(v) := \mathcal{N}(v) := 0$;
29. $\text{ГРУППА}(v) :=$ пустой список;
30. $Id(v) := 0$;
31. $\mathcal{L}(v) := n$;
32. **конец** цикла;
33. $\text{ПОИСК}(s)$;
34. $n_1 := \bar{m}$;

примечание отныне мы полагаем, что каждая вершина идентифицируется своим M -номером **конец** примечания

35. модифицируем все структуры данных так, чтобы все вершины идентифицировались своими номерами;

Шаг 2

36. **для** i от 1 до n , **шаг 1** **цикл**

примечание производится вычисление $H(v)$ для каждой достижимой вершины v **конец** примечания

37. **начало**
38. $H(i) := 0$;
39. $\sigma(i) :=$ пустое множество;
40. **конец**
41. **для** i от 2 до n , **шаг 1** **цикл**
42. $\sigma(\text{ПРЕДОК}(i)) := \sigma(\text{ПРЕДОК}(i) \cup \sigma(i))$;

43. для w от n_1 до 1 шаг — 1 цикл

44. начало

45. пока ГРУППА(w) $\neq \emptyset$ цикл

46. начало

47. выбрать (u, w) из ГРУППА(w);

48. удалить (u, w) из ГРУППА(w);

49. пока $\neg(u \xrightarrow{*} w)$ цикл

примечание запись $u \xrightarrow{*} w$ означает, что w достижима из u по древесным дугам **конец примечания**

50. начало

51. $x := \text{НАЙТИ}(u)$;

примечание функция НАЙТИ(x) вычисляет номер множества, содержащего x в качестве элемента **конец примечания**

52. если $H(u) = 0$ то

53. начало

54. $\sigma(x) := \sigma(x) \cup \sigma(u)$;

55. $H(u) := w$;

56. **конец**

примечание после этого шага все величины $H(v)$ будут вычислены правильно **конец примечания**

57. $u := x$;

58. **конец цикла**;

59. **конец цикла**;

60. **конец цикла**;

Шаг 3

61. примечание на этом шаге производится вычисление семидоминаторов **конец примечания**

62. для i от 1 до n_1 шаг 1 цикл

63. $\sigma_1(i) := \{i\}$;

примечание σ_1 содержит информацию о поддеревьях доминаторного дерева **конец примечания**

64. для i от n_1 до 1 шаг — 1 цикл

65. начало

66. для каждой поперечной дуги (u, i) цикл

67. начало

примечание производится превращение дуги (u, i) в прямую дугу повторяющейся заменой поперечных дуг **конец примечания**

68. $x := \text{НАЙТИ}(u)$;

69. если $(x \xrightarrow{*} i)$ то

70. заменить (u, i) на (x, i)

71. иначе начало

72. $(x, v) := \text{НАЙТИ}(x)$;

73. заменить (u, i) на (x, i) ;

74. конец

75. если $x \xrightarrow{*} \mathcal{L}(i)$ то $\mathcal{L}(i) := x$;

примечание если в i заходят две прямые дуги, то одна удаляется конец примечания

76. конец цикла;

77. если $H(v) < v \wedge \mathcal{L}(v) < v \wedge \mathcal{L}(\mathcal{L}(v)) > H(v)$ то

78. $\mathcal{L}(\mathcal{L}(v)) := H(v)$;

примечание если из i исходит обратная дуга, то она удаляется и добавляется, если необходимо, прямая дуга конец примечания

79. примечание вычисляются $Sd(i-1, v)$ для всех таких v , что $Sd(i-1, v) \neq Sd(i, v)$; семидоминатор вершины v есть приоритет σ_2 , содержащего ее, если $v \geq 1$ и $Id(v)$ пока еще не известны конец примечания

80. ОЧЕРЕДЬ(i) := пустая очередь;

81. для $w \in \Gamma_i$ цикл

82. начало

83. ОЧЕРЕДЬ(i) := ОЧЕРЕДЬ(i) \cup ОЧЕРЕДЬ(\bar{w});

84. удалить $\sigma_2(z, j)$ с наивысшим приоритетом z из ОЧЕРЕДЬ(i);

85. конец цикла;

86. пока $z = i$ цикл

87. начало

88. для каждого элемента $v \in \sigma_2(z, j)$ цикл

89. начало

90. $Id(v) := i$;

91. $\sigma_1(i) := \sigma_1(i) \cup \sigma_1(v)$;

92. конец цикла;

93. удалить $\sigma_2(z, j)$ с наивысшим приоритетом z из ОЧЕРЕДЬ(i);

94. конец цикла;

95. добавить $\sigma_2(z, j)$ к ОЧЕРЕДЬ(i);

96. если $\mathcal{L}(i) \geq i$ то

97. начало

98. $\sigma_2(\text{ПРЕДОК}(i), i) := \{i\}$;

99. добавить $\sigma_2(\text{ПРЕДОК}(i), i)$ к ОЧЕРЕДЬ(i);

100. конец цикла;

101. иначе начало

102. $\sigma_2(\mathcal{L}(i), i) := \{i\}$;

103. удалить $\sigma_2(z, j)$ с наивысшим приоритетом z из ОЧЕРЕДЬ(i);

104. конец

105. пока $\mathcal{L}(i) \leq z$ цикл
106. начало
107. $\sigma_2(\mathcal{L}(i), i) := \sigma_2(\mathcal{L}(i), i) \cup \sigma_2(z, j)$;
108. удалить $\sigma_2(z, j)$ с наивысшим приоритетом z
из ОЧЕРЕДЬ(i);
109. конец цикла;
110. добавить $\sigma_2(z, j)$ к ОЧЕРЕДЬ(i);
111. добавить $\sigma_2(\mathcal{L}(i), i)$ к ОЧЕРЕДЬ(i);
112. конец цикла;
- конец

4.4. Замечания к алгоритму. 1. Шаг 1 имеет трудоемкость $O(n + m)$.

2. Прямой алгоритм для вычисления $H(v)$ требует $O(n^2)$ времени. В данном случае предполагается использование хорошего алгоритма для вычисления объединений множеств, в результате чего алгоритм имеет почти линейную трудоемкость. Обратные дуги вначале рассортировываются с помощью M -номеров их концов, потом вычисляются $H(v)$ с помощью обработки обратных дуг (u, w) в порядке убывания номеров вершин w . Каждая вершина получает метку H точно один раз. Если (u, w) — следующая для обработки обратная дуга, то каждая не помеченная к данному моменту вершина, исключая w , на пути из w в u имеет метку H , равную w , и может быть так помечена.

3. После строки 5б все значения H будут вычислены. Чтобы алгоритм работал эффективно, шаги 4з—5б алгоритма не должны перепроверять, вычислены или нет значения H для вершин. Для этого нужно иметь хороший метод для вычисления объединения множеств. У нас будут подмножества, занумерованные от 1 до n . Всякая вершина $v \neq 1$ всегда будет находиться в множестве, номер которого равен наибольшему собственному непомеченному предку вершины v ; вначале, если (v, w) — древесная дуга, вершина w входит в множество, имя которого v .

Чтобы обработать обратную дугу (u, w) , найдем множество u_1 , содержащее u , множество u_2 , содержащее u_1 , и т. д., пока не достигнем множества u_n такого, что $u_n \rightarrow w$. Вершины u_1, u_2, \dots, u_{n-1} и, возможно, u суть непомеченные вершины на пути в дереве T из w в u . Мы пометим их значением H вершины w и потом вычислим объединение множеств с именами $u_1, u_2, \dots, u_{n-1}, u_n$ и, возможно, u , дав ему имя u_n .

Все объединения множеств здесь являются объединениями непересекающихся множеств. Операция $\text{НАЙТИ}(x)$ вычисляет номер множества, содержащего x в качестве элемента. При использовании эффективных реализаций для поиска и объединения множеств трудоемкость шага 2 есть $O(n \log n + m)$.

4. Назовем дерево *доминаторным деревом* для графа G , если оно имеет то же множество вершин, что и G , и вершина w является потомком вершины v в дереве тогда и только тогда, когда v есть непосредственный доминатор для w . Предками вершины v в дереве являются в точности все доминаторы для w .

При вычислении семидоминаторов на шаге 3 строится доминаторное дерево от висячих вершин к корню: в любой данный момент времени часть доминаторного дерева, которую мы знаем, будет состоять из отдельных вершин и их потомков. Пусть это множество поддеревьев есть F . Для хранения информации о F используются множества σ_i , занумерованные от 1 до n_i . Если v — вершина в F , то v будет храниться в множестве σ_i , номер которого есть M -номер корня того поддерева F , которое содержит v . Если v не входит ни в одно поддерево F , то v будет храниться в $\sigma_1(v)$. Первоначально каждое множество $\sigma(v)$ содержит только v . Для изменения множества σ_i каждый раз, когда мы вычисляем $Id(v)$ для новой вершины, мы полагаем $\sigma_i(Id(v)) = \sigma_i(v) \cup \sigma_i(Id(v))$. Для этого можно использовать эффективный алгоритм объединения множеств.

Так как для большинства вершин $Sd(i-1, v) = Sd(i, v)$, то имеет смысл вычислять только те семидоминаторы, которые изменяются при изменении i . В качестве структур данных для этих вычислений используются очереди с приоритетами.

Чтобы реализовать вычисления семидоминаторов, используя приоритетные очереди, создадим очередь для каждой вершины v . Очередь для вершины v будет содержать множества σ_2 , каждое из которых есть множество потомков v . Все вершины в σ_2 будут иметь одно и то же значение $Sd(v, w)$, и это значение будет приоритетом множества σ_2 в очереди. В множество σ_2 включаются только те вершины, чьи значения Id не известны. Таким образом, если $Sd(v, w)$ есть приоритет некоторого множества σ_2 в очереди для вершины v , то имеет место соотношение $Sd(v, w) \xrightarrow{*} v$.

Чтобы модифицировать семидоминаторы при обработке вершины v , обозначим через $(\mathcal{L}(v), v)$ прямую дугу (произвольную, если их несколько), заходящую в v . Все вершины из Γv к этому моменту уже обработаны. Сначала мы конструируем приоритетную очередь для v , комбинируя очереди вершин из Γv . Каждое множество σ_2 в новой очереди имеет приоритет, соответствующий некоторому предку вершины v . Удалим каждое множество σ_2 , имеющее приоритет v — наивысший возможный приоритет. Каждая вершина в удаленном множестве σ_2 имеет значение Id , равное v (по лемме 14). Пометим эти вершины значениями Id и модифицируем множества σ_1 . Если v не имеет входящих в нее прямых дуг, то добавим к очереди новое множество $\sigma_2(v)$ с приоритетом, равным непосредственному предку вершины v . Если в v входит прямая дуга, то удалим из очереди каждое множество σ_2 с приоритетом, равным или большим чем $\mathcal{L}(v)$, найдем объединение σ_2 всех этих множеств, добавим к нему в качестве элемента вершину v и включим это новое множество σ_2 в очередь с приоритетом $\mathcal{L}(v)$. Это реализует лемму 18 для модификации семидоминаторов.

Библиографический комментарий

Алгоритм Килдала был опубликован в 1973 г. в работе [122], изложенный в § 1 вариант алгоритма Килдала принадлежит Дж. Каму и Дж. Ульману [120] (см. также [11]). Он основан на поиске в глубину.

Кратчайшим путям посвящена обширная литература, наиболее распространенные методы Форда, Дейкстры и др. изложены во многих книгах по прикладным аспектам теории графов, например в [7, 9, 72, 78] и др. Методы отыскания путей при наличии ограничений можно найти в [27]. Алгоритм отыскания путей между заданными вершинами в обыкновенном графе принадлежит С. Межику [135], во взвешенном графе — С. Цукняме и др. [157]. По поводу генерирования всех путей в графе с использованием матрицы смежности см. [52].

Описанный в § 3 метод отыскания кратчайшего пути, проходящего через заданное множество вершин, принадлежит Х. Габову и др. [114] (о других методах и подходах см. [27]). Задачи о покрытиях вершин графа излагаются по работе С. Нтафоса и С. Хакими [141], метод построения приведенного покрытия дуг орграфа — по работе И. С. Грунского и Д. В. Сперанского [22]. Теорема 3 из п. 3.2 известна в комбинаторике под названием теоремы Дилуорса [81]. О потоках в сети см. [1, 9, 78]. Упомянутый в п. 3.4 алгоритм для решения задачи о максимальном паросочетании принадлежит Дж. Хопкрофту и Р. Карпу (об этом см. [53]). Отыскание доминаторов в орграфе излагается по работе Р. Тарьяна [153]; об эффективном алгоритме построения объединения непересекающихся множеств см. [7].

ГЛАВА 4

ОПТИМИЗАЦИОННЫЕ ЗАДАЧИ НА ГРАФАХ

§ 1. Построение оптимальных нумераций

1.1. Нумерацией, *оптимальной* относительно функционала $\mathcal{F}(F)$, называется такая нумерация F вершин графа, для которой значение функционала \mathcal{F} минимально. Оптимальная нумерация F называется *допустимой*, если для любой дуги (x, y) графа имеет место неравенство $F(x) > F(y)$. Допустимая нумерация F называется *плоской*, если при расположении вершин графа в целочисленных точках числовой оси его дуги не пересекаются.

В дальнейшем мы будем рассматривать только допустимые нумерации. Кроме того, мы будем предполагать, что вершины орграфа расположены в целочисленных точках числовой оси. В силу этого соглашения будем говорить не о нумерации вершин графа, а об укладке графа в одномерную целочисленную решетку или просто об укладке. Укладку F будем записывать в виде $F = (v_1, \dots, v_n)$, причем имеет место цепочка неравенств $F(v_1) < F(v_2) < \dots < F(v_n)$.

Растянутостью укладки $F(v_1, \dots, v_n)$ называется число $P_F = \max(F(v_1), \dots, F(v_n)) - \min(F(v_1), \dots, F(v_n)) - n + 1$. *Сжатием* укладки F на множество чисел $a + 1, \dots, a + n$, где a — натуральное число, называется укладка F_1 , у которой $F_1(v_1) = a + 1, \dots, F_1(v_n) = a + n$.

Заметим, что при сжатии укладки допустимость нумерации сохраняется.

Пусть дана укладка $F(v_1, \dots, v_n)$. Будем говорить, что дуга (x, y) проходит над вершиной z , если имеет место неравенство $F(y) < F(z) < F(x)$. Дуги (x, y) и

(v, w) пересекаются, если выполняется одно из условий:

а) дуга (x, y) проходит над вершиной v , и дуга (v, w) проходит над вершиной y ;

б) дуга (x, y) проходит над вершиной w , и дуга (v, w) проходит над вершиной x .

Пусть $w(x)$ есть число дуг, проходящих над вершиной x .

Шириной укладки F называется функционал $W(F) = \max_x w(x)$. Укладка, ширина которой минимальна среди всех упаковок графа, называется *оптимальной по ширине* упаковкой, а ее ширина — *шириной* графа.

Длиной укладки $F(v_1, \dots, v_n)$ называется функционал $H(F) = \sum_{(x,y) \in E(G)} |F(x) - F(y)|$. Укладка, длина которой минимальна среди всех упаковок данного графа, называется *оптимальной по длине* упаковкой, а ее длина — *длиной* графа.

1.2. Построение оптимальной по ширине укладки. Пусть $T = (V, E)$ — корневое ордеревое с корнем r . Укладка минимальной ширины для дерева T получается в результате заполнения таблицы с тремя столбцами и n строками, где $n = |V|$, по строке на каждую вершину, и пусть $\rho(i, j)$ обозначает содержимое клетки таблицы на пересечении i -й строки и j -го столбца, $i = 1, \dots, n$; $j = 1, 2, 3$.

Алгоритм:

Вход: Ордеревое $T = (V, E)$ с корнем r , заданное списками смежности.

Выход: Оптимальная по ширине укладка $\varphi = (v_1, \dots, v_n)$ и ширина $W(T)$ дерева T .

начало

1. провести поиск в ширину и занумеровать вершины в соответствии с порядком их обхода при этом поиске;

примечание это означает, что номер 1 получает корень, далее нумеруются его потомки, затем потомки потомков и т. д. **конец примечания**

2. для i от n до 1 шаг —1 цикл

3. **начало**

4. если номер вершины v равен i то $\rho(i, 1) := v$;

примечание с этого момента вершины идентифи-

цируются номерами соответствующих строк таблицы
конец примечания

5. **конец** цикла;

6. для i от 1 пока $i \leq n$ цикл

7. **начало**

8. если i — височная вершина то

9. **начало**

10. $\rho(i, 2) := 0$;

11. $\rho(i, 3) := i$;

12. **конец**

13. иначе **начало**

14. упорядочить вершины y_0, y_1, \dots, y_k из Γ_i в порядке убывания содержимого $\rho(y, 2)$ второго столбца, пусть упорядоченная последовательность есть $y_{j_0}, y_{j_1}, \dots, y_{j_k}$;

примечание в соответствии с правилами заполнения первого столбца все строки, соответствующие вершинам $y_{j_0}, y_{j_1}, \dots, y_{j_k}$, уже заполнены **конец** примечания

15. для t от 0 до k шаг 1 цикл

16. **начало**

17. $p_t := \rho(y_{j_k}, 2) + t$;

18. $\rho(i, 2) := \max(\rho(i, 2), p_t)$;

19. $\rho(i, 3) := \rho(i, 3) \rho(y_{j_t}, 3)$;

20. **конец** цикла;

21. **конец**;

22. **конец** цикла;

23. $w := \rho(n, 2)$;

24. $\varphi := \rho(n, 3)$;

конец

Пример заполнения таблицы для дерева T , показанного на рис. 4.1, а, приведен на рис. 4.1, б.

Предположим теперь, что имеется таблица, заполненная по алгоритму для дерева $T = (V, E)$ с корнем r .

Теорема 1. Для каждой вершины $v \in V$ содержимое третьей клетки строки, соответствующей вершине v , т. е. $\rho(v, 3)$, есть укладка поддеревы $T(v)$ с корнем в вершине v ; при этом ее ширина равна числу, стоящему во второй клетке той же строки, т. е. $\rho(v, 2)$.

Из теоремы и того факта, что $T(r) = T$, немедленно следует, что последовательность $\varphi(r)$, стоящая в по-

следней строке третьего столбца, есть укладка дерева T с шириной $W = \rho(r, 2)$.

Теорема 2. Для каждой укладки дерева T ее ширина не меньше содержимого $\rho(r, 2)$ второй клетки последней строки.

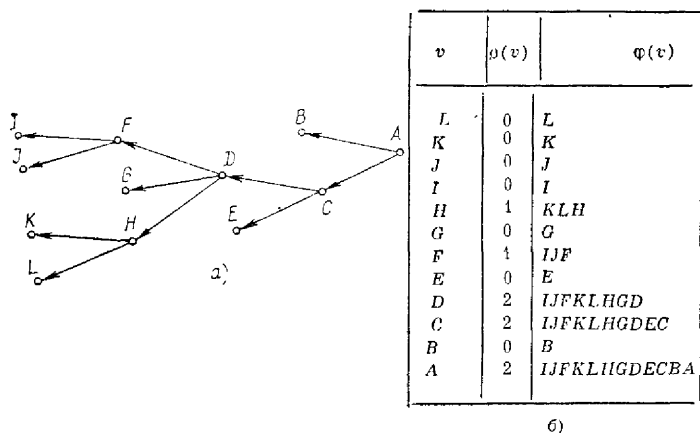


Рис. 4.1.

Из теоремы 2 окончательно вытекает, что алгоритм действительно строит укладку минимальной ширины.

1.3. Построение оптимальной по ширине укладки взвешенного ордерова. Пусть каждой дуге $u \in E$ ордерова $T = (V, E)$ приписан неотрицательный вес $P(u)$, а каждой вершине v — вес $P(v)$. Рассмотрим плоскую укладку ϕ взвешенного ордерова T . Определим для каждой вершины v полный вес $Q(v)$ следующим образом:

— если вершина v — висячая, то $Q(v) = P(v)$;

— если вершина v — внутренняя с множеством потомков $\Gamma v = \{y_1, \dots, y_m\}$, причем вершины y_i выписаны в порядке их следования в плоской укладке ϕ , то

$$Q(v) = \max \left\{ \max_{1 \leq j \leq m} \left[\sum_{i=1}^{j-1} R(y_i) + Q(y_i) \right], \sum_{i=1}^m R(y_i) + P(v) \right\}, \quad (4.1)$$

где $R(y_i)$ есть вес дуги, заходящей в вершину y_i .

Будем считать, что $R(v) \leq \sum_{i=1}^m R(y_i) + P(v)$; тогда имеет место неравенство $Q(v) - R(v) \geq 0$.

Шириной плоской укладки $\varphi(v)$ взвешенного под-
дерева $T(v)$ с корнем v называется полный вес $Q(v)$
вершины v . (В случае $v=r$ это будет служить опре-
делением ширины плоской укладки всего дерева.)

Изложим метод решения задачи о построении ук-
ладки с наименьшей шириной. Вначале рассмотрим
решение в классе плоских упаковок, а затем в произ-
вольном случае.

Теорема 3. *Для того чтобы полный вес внутрен-
ней вершины v в классе плоских упаковок $\Phi_{\text{пл}}$ был
минимален, достаточно упорядочить ее потомков y_1, \dots
 \dots, y_m согласно соотношению*

$$Q(y_1) - R(y_1) \geq Q(y_2) - R(y_2) \geq \dots \geq Q(y_m) - R(y_m). \quad (4.2)$$

Для доказательства теоремы необходима следующая
лемма.

Лемма 1. *Укладка $\varphi_1 = (v_1, \dots, v_k, v_{k+1}, \dots, v_m)$
потомков вершины обеспечивает полный вес вершины
 v не больший, чем укладка $\varphi_2 = (v_1, \dots, v_{k+1}, v_k, \dots, v_m)$,
если*

$$Q(v_k) - R(v_k) \geq Q(v_{k+1}) - R(v_{k+1}). \quad (4.3)$$

Рассмотрим три случая.

1. Допустим, что $Q^{(1)}(v) = \sum_{i=1}^{j-1} R(v_i) + Q(v_j)$, $j \neq k$,
 $k+1$, или $Q^{(1)}(v) = \sum_{i=1}^m R(v_i) + P(v)$. Тогда $Q^{(2)}(v) \geq$
 $\geq Q^{(1)}(v)$.

2. Если $Q^{(1)}(v) = \sum_{i=1}^k R(v_i) + (Q(v_k) - R(v_k))$, то
 $Q^{(2)}(v) \geq \sum_{i=1}^{k+1} R(v_i) + (Q(v_k) - R(v_k)) \geq$
 $\geq \sum_{i=1}^k R(v_i) + (Q(v_k) - R(v_k)) \geq Q^{(1)}(v)$.

3. При $Q^{(1)}(v) = \sum_{i=1}^{k+1} R(v_i) + (Q(v_{k+1}) - R(v_{k+1}))$ имеем,
используя (4.3),

$$Q^{(2)}(v) \geq \sum_{i=1}^{k+1} R(v_i) + (Q(v_k) - R(v_k)) \geq$$

$$\geq \sum_{i=1}^{k+1} R(v_i) + (Q(v_{k+1}) - R(v_{k+1})) = Q^{(1)}(v).$$

Следовательно, при всех возможных случаях справедливо неравенство

$$Q^{(2)}(v) \geq Q^{(1)}(v).$$

Доказательство теоремы 3. Пусть для некоторой укладки Φ потомков вершины v выполнены соотношения (4.2). Предположим, что Φ неоптимальна, т. е. существует укладка Φ' , обеспечивающая минимальный вес $Q(v)$. Так как $\Phi' \neq \Phi$, то в Φ' найдется хотя бы одна пара соседних вершин v_i и v_{i+1} , таких, что $Q(v_i) - R(v_i) < Q(v_{i+1}) - R(v_{i+1})$. Согласно лемме 1, поменяв местами эти вершины, мы получим улучшение полного веса $Q(v)$, что противоречит оптимальности Φ' . Следовательно, укладка Φ с соотношениями (4.2) обеспечивает минимум полного веса $Q(v)$.

Из теоремы 3 непосредственно вытекает алгоритм нахождения в классе $\Phi_{пл}$ плоской укладки минимальной ширины. Он заключается в том, что если на каком-то шаге для внутренней вершины $v \in V$ известны минимальные полные веса ее потомков, то для получения минимального полного веса $Q(v)$ вершины v необходимо упорядочить ее потомков согласно соотношениям (4.2). Процесс заканчивается определением минимального веса корня с одновременным получением плоской укладки минимальной ширины всего дерева.

Рассмотрим решение поставленной задачи в классе произвольных упаковок. Пусть задана произвольная укладка Φ ордерова T . Будем считать, что Φ состоит из нескольких последовательно расположенных кусков ψ_1, \dots, ψ_m , каждый из которых включает набор вершин, расположенных в порядке их следования в укладке Φ . Обозначим через $\mu(v)$ путь из корня r дерева в вершину v . Рассмотрим вершину v_{j_i} из ψ_j . Пусть v_{t_i} — первая из вершин ψ_i , лежащая на пути из r в ψ_i и такая, что $v_{j_i} \in \mu(v_{t_i})$, $i < j$, $1 < j \leq m$.

Переопределим вес вершины v_{j_i} (новые характеристики помечены штрихом):

$$R'(v_{j_i}) = R(v_{j_i}) - \sum_{i=1}^{j-1} R(v_{t_i}),$$

и определим вес куска ψ_j и полный вес куска ψ_j

В итоге получаем соотношения

$$Q(\psi'_1) - R(\psi'_1) \geq Q(\psi'_2) - R(\psi'_2) \geq \dots \\ \dots \geq Q(\psi'_q) - R(\psi'_q).$$

Отсюда и из теоремы 3 вытекает, что куски ψ'_1, \dots, ψ'_q должны следовать в минимальной укладке дерева в том же порядке, в каком они следуют в минимальной укладке поддерева.

Такое же разбиение проводим над укладками всех остальных поддеревьев $\sigma_2, \sigma_3, \dots, \sigma_l$.

Возьмем произвольную укладку φ дерева T . Будем последовательно улучшать φ , приведя ее к минимальной. Закрепим положение вершины v'_α в укладке φ и стянем к ней слева и справа все вершины, которые принадлежат куску ψ'_1 , одновременно упорядочивая их согласно следовавшей в укладке α_1 . При этом положение вершин, стянутых справа, не ухудшится, а из того, что $R(\psi'_1)$ минимально, вытекает, что и положение вершин, следующих за ними, также не ухудшится. Так как на вершине v'_α достигается максимум в ψ'_1 , то, хотя положение вершин из ψ'_1 , стянутых слева, и ухудшается, ширина укладки в них не больше ширины укладки в v'_α . Положение оставшихся слева вершин не ухудшится. Такую операцию проведем со всеми кусками всех поддеревьев. После этого упорядочим сформированные куски по убыванию разностей $Q(\psi_j) - R(\psi_j)$, $j = 1, 2, \dots$

Единственность получаемой таким образом укладки следует из ее построения. Следовательно, она будет и минимальной ввиду произвольности исходной укладки.

Следствие. Для того чтобы минимальная укладка дерева была плоской, достаточно, чтобы для любой внутренней вершины $v \in V$ и любого ее потомка y выполнялось неравенство $R(v) \leq R(y)$.

А л г о р и т м:

Вход: Дерево $T = (V, E)$, веса дуг и веса вершин.

Выход: Минимальная укладка.

начало

1. занумеровать все вершины дерева в соответствии с некоторой произвольной укладкой $\varphi = (v_1, \dots, v_n)$ дерева;

2. организовать информацию о дереве в виде таблицы M_1 :

v_1	$\Gamma^{-1}v_1$	$R(v_1)$	$P(v_1)$
v_2	$\Gamma^{-1}v_2$	$R(v_2)$	$P(v_2)$
...			
v_{n-1}	$\Gamma^{-1}v_{n-1}$	$R(v_{n-1})$	$P(v_{n-1})$
v_n		$R(v_n)$	$P(v_n)$

3. для каждого множества Γv_i , у которого элементы в M_1 расположены последовательно, цикл

4. начало

5. провести разбиение каждого множества вершин $\{\Gamma v_i, v_i\}$ на минимальные куски способом, изложенным в доказательстве теоремы 4;

6. определить характеристики кусков $R(\psi_i)$ и $Q(\psi_i)$;

7. конец цикла;

8. сформировать таблицу M_2 , в которой каждая запись соответствует куску, а не вершине и имеет вид:

ψ_i	$\Gamma^{-1}\psi_i$	$R(\psi_i)$	$Q(\psi_i)$
----------	---------------------	-------------	-------------

9. если все минимальные куски приписаны к корню то стоп

10. иначе переход к шагу 3;

Ясно, что последний сформированный массив определит минимальную укладку дерева.

Пример. Рассмотрим дерево на рис. 4.2; его характеристики заданы таблицей 4.1.

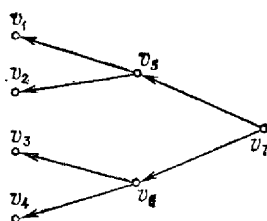


Рис. 4.2.

Таблица 4.1

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
R	5	1	5	5	10	10	—
P	20	1	7	6	4	5	1
Q	20	1	7	6	20	15	25

Полные веса вершин в этой таблице соответствуют плоской укладке минимальной ширины: $\varphi = (v_1, v_2, v_5, v_3, v_4, v_6, v_7)$. Ее ширина равна $Q = 25$. Составим таблицу M_1 (таблица 4.2).

Т а б л и ц а 4.2

v_1	v_4	5	20
v_2	v_5	1	1
v_5	v_7	10	4
v_3	v_6	5	7
v_4	v_6	5	6
v_6	v_7	10	5
v_7	v_7	—	1

Просматриваем таблицу M_1 . Находим множество $\{v_1, v_2\}$, приписанное к вершине v_5 . Рассмотрим множество $\{v_1, v_2, v_5\}$. Для него $\max\{20; 5+1; 5+1+4\} = 20$, т. е. максимум достигается на v и $R_1 < R_5$. образуем первый кусок, состоящий из v_1 : $\psi_1 = (v_1)$; второй кусок, состоящий из вершин v_2 и v_5 : $\psi_2 = (v_2, v_5)$ (его параметры следующие: $Q(\psi_2) = 5$, $R(\psi_2) = R_5 - R_1 = 5$). Третий кусок образуется из множества $\{v_3, v_4\}$, приписанного к вершине v_6 . Для расширенного множества $\{v_3, v_4, v_6\}$ имеем $\max\{7; 5+6; 5+5+5\} = 15$, и, так как максимум достигается на вершине v_6 , третий кусок имеет вид $\psi_3 = (v_3, v_4, v_6)$; его параметры: $Q(\psi_3) = 15$, $R(\psi_3) = 10$. Все три куска приписаны к вершине v_7 . Составим таблицу M_2 (таблица 4.3).

Т а б л и ц а 4.3

ψ_1	v_7	5	20
ψ_2	v_7	5	5
ψ_3	v_7	10	15
ψ_4	v_7	—	1

В таблице M_2 все куски приписаны к вершине v_7 , являющейся корнем. Алгоритм заканчивается. Для

получения минимальной укладки упорядочим куски в величине разностей $Q - R$. Итоговая последовательность $(\psi_1, \psi_3, \psi_2, \psi_4)$, для которой $\max [20; 5 + 15; 5 + 10 + 5; 5 + 10 + 5 + 1] = 21$, определяет минимальную укладку $\varphi_0 = (v_1, v_3, v_4, v_6, v_2, v_5, v_7)$ с шириной 21.

1.4. Построение оптимальной по длине укладки ор-дерова.

Лемма 2. Пусть $T = (V, E)$ — дерево с n вершинами, φ — нумерация его вершин с растянутостью $P_\varphi(V) = k$, φ_1 — сжатие φ на множество номеров $\{1, 2, \dots, n\}$. Имеет место неравенство $\Pi(T, \varphi_1) \leq \Pi(T, \varphi) - k$.

Доказательство. Если растянутость нумерации φ не равна нулю, то это означает, что между отдельными группами номеров, идущих подряд, имеются пробелы. Будем строить новую нумерацию, выделив в φ последнюю группу подряд идущих k_1 пробелов, сохранив номера с первого до стоящего перед последней группой пробелов и уменьшив остальные номера на k_1 . Тем самым множество вершин разбивается на два подмножества — V_1 с измененными номерами и V_2 с оставшимися неизменными номерами. Длины дуг $|\varphi(x) - \varphi(y)|$ дерева уменьшаются на k_1 только в том случае, если они соединяют вершины из V_1 и V_2 , а такая дуга, по крайней мере одна, существует. Продолжая процесс, получим искомый результат. Лемма доказана.

Пусть $T = (V, E)$ — произвольное дерево с n вершинами, Φ_n — совокупность всех допустимых нумераций φ дерева T , таких, что $\varphi(V) = \{1, \dots, n\}$. Из леммы 2 следует, что в Φ_n обязательно содержится оптимальная нумерация дерева T , поскольку для любой нумерации φ_1 дерева T нумерация φ_0 , являющаяся сжатием φ_1 на множество номеров $\{1, 2, \dots, n\}$, а значит, принадлежащая Φ_n , обладает свойством $\Pi(T, \varphi_1) \leq \Pi(T, \varphi_0) - P_{\varphi_0}(V) \leq \Pi(T, \varphi_0)$. В оставшейся части параграфа под оптимальной нумерацией произвольного дерева T с n вершинами будет подразумеваться именно оптимальная нумерация, содержащаяся в Φ_n .

Пусть вершина v имеет своими потомками вершины y_1, y_2, \dots, y_d ($d > 0$), и пусть n_i есть количество вершин в поддереве $T(y_i)$ с корнем y_i , $i \in [1 : d]$. Можно считать вершины y_1, \dots, y_d упорядоченными так, что $n_1 \geq n_2 \geq \dots \geq n_d$.

Назовем *весом* вершины v величину $b(v)$, где

$$b(v) = \begin{cases} \sum_{i=1}^d ((i-1)n_i + 1), & \text{если } d \neq 0; \\ 0 & \text{в противном случае.} \end{cases}$$

Построим нумерацию $A(T)$ дерева T , обладающую следующими свойствами:

- а) $A(T)$ — допустимая нумерация дерева T ;
- б) если v — корень дерева T , y_1, \dots, y_d — его по-

томки и $\varphi = A(T)$, то $\Pi(T, \varphi) = b(v) + \sum_{i=1}^d \Pi(T(y_i))$;

- в) $A(T)$ — оптимальная нумерация дерева T .

Построение нумерации $A(T)$ и доказательств свойств а) — в) проводятся индукцией по количеству вершин дерева.

Дерево T_0 , состоящее из единственной вершины v_0 , имеет тривиальную оптимальную нумерацию φ_0 : $\varphi_0(v_0) = 1$. Положим $A(T_0) = \varphi_0$. Пусть $T = (V, E)$ — n -вершинное дерево с корнем v_0 ; v_1, \dots, v_d — его потомки, $d \geq 1$, n_i — число вершин в поддереве $T(v_i)$, $i \in [1:d]$. Тогда

$$n = \sum_{i=1}^d n_i + 1. \quad (4.4)$$

Можно считать, что $n_1 \geq n_2 \geq \dots \geq n_d$. Каждое из поддеревьев $T(v_i)$ имеет меньше чем n вершин, а значит, по предположению индукции нумерации $A(T(v_i))$ определены и обладают свойствами а) — в). Пусть $\varphi_i = A(T(v_i))$, $i \in [1:d]$. Образует φ'_2 — сжатие φ_2 на множество $n_1 + 1, n_1 + 2, \dots, n_1 + n_2$ (фактически φ'_2 является сдвигом φ_2); φ'_3 — сжатие φ_3 на множество $n_1 + n_2 + 1, \dots, n_1 + n_2 + n_3$; \dots , φ'_k — сжатие φ_k на множество

$$\sum_{i=1}^{k-1} n_i + 1, \quad \sum_{i=1}^{k-1} n_i + 2, \quad \dots, \quad \sum_{i=1}^{k-1} n_i + n_k; \quad k = 1, \dots, d_2$$

и φ'_d — сжатие φ_d на множество

$$\sum_{i=1}^{d-1} n_i + 1, \quad \sum_{i=1}^{d-1} n_i + 2, \quad \dots, \quad \sum_{i=1}^{d-1} n_i + n_d.$$

Каждая вершина дерева T , кроме v_0 , принадлежит одному из поддеревьев $T(v_i)$, $i \in [1:d]$, и соответству-

ющей нумерацией ей сопоставлено некоторое число между 1 и $\sum_{i=1}^d n_i = n - 1$, причем разным вершинам сопоставлены разные числа. Вершине v_0 сопоставим число n . Теперь всем вершинам дерева T сопоставлены различные номера от 1 до n , а значит, мы построили нумерацию $A(T)$ для дерева T . Обозначим ее через φ_0 .

Покажем, что φ_0 обладает свойством а). Возьмем произвольную дугу $(x, y) \in E$. Если $x = v_0$, то $\varphi_0(x) = n$, и, следовательно, $\varphi_0(y) \leq n - 1 < n$. Если же $x \neq v_0$, то $(x, y) \in T(v_i)$ для некоторого i между 1 и d ; а так как φ_i по предположению индукции допустима, φ'_i есть сжатие φ_i , то φ'_i допустима и $\varphi_0(x) = \varphi'_i(x) > \varphi'_i(y) = \varphi_0(y)$.

Покажем, что φ_0 обладает свойством б). Очевидно, что

$$\Pi(T, \varphi_0) = \sum_{i=1}^d \Pi(T(v_i), \varphi'_i) + \sum_{i=1}^d (\varphi_0(v_0) - \varphi_0(v_i)). \quad (4.5)$$

При переходе от φ_i к φ'_i номера всех вершин в $T(v_i)$ изменяются на одно и то же число, поэтому из оптимальности φ'_i для $T(v_i)$ следует

$$\Pi(T(v_i), \varphi'_i) = \Pi(T(v_i), \varphi_i) = \Pi(T(v_i)). \quad (4.6)$$

Так как вершина v_i — корень поддерева $T(v_i)$, то ее номер является наибольшим среди номеров вершин поддерева $T(v_i)$, т. е. $\varphi_0(v_i) = \sum_{k=1}^i n_k$ для всех i . Подставляя это равенство вместе с (4.6) в (4.5) и учитывая (4.4), получаем

$$\begin{aligned} \Pi(T, \varphi_0) &= \sum_{i=1}^d \Pi(T(v_i)) + \sum_{i=1}^d \left(\sum_{k=1}^d n_k + 1 - \sum_{k=1}^i n_k \right) = \\ &= \sum_{i=1}^d \Pi(T(v_i)) + \sum_{i=1}^d \left(\sum_{k=i+1}^d n_k + 1 \right) = \\ &= \sum_{i=1}^d \Pi(T(v_i)) + n_2 + 2n_3 + 3n_4 + \dots + (d-1)n_d + d_1 \quad (4.7) \end{aligned}$$

$$\text{т. е. } \Pi(T, \varphi_0) = b(v_0) + \sum_{i=1}^d \Pi(T(v_i)).$$

Покажем, что φ , обладает свойством в). Возможны два случая: $\deg^+(v_0) = d = 1$ и $\deg^+(v_0) = d > 1$. В первом случае для любой допустимой нумерации φ дерева T имеем $\varphi(v_0) = n$ и $\varphi(v_1) = n - 1$. Обозначив через ψ нумерацию, которую φ индуцирует на $T(v_1)$, получаем, что

$$\begin{aligned} \Pi(T, \varphi) &= \Pi(T(v_1), \psi) + 1 \geq \Pi(T(v_1), \varphi_1) + 1 = \Pi(T, \varphi_0), \\ \text{т. е. что } \varphi_0 &\text{ оптимальна. Рассмотрим второй случай. Пусть } \varphi \text{ — произвольная нумерация вершин дерева } T, \\ \text{и пусть } \varphi^{-1}(1) &\in T(v_k), \text{ т. е. вершина, имеющая при нумерации } \varphi \text{ номер } 1, \text{ принадлежит поддереву } T(v_k). \\ \text{Обозначим через } V_k &\text{ множество вершин дерева } T(v_k), \text{ через } \bar{T}(v_k) \text{ — поддерево дерева } T, \text{ порожденное мно-} \\ \text{жеством вершин } \bar{V}_k &= V \setminus V_k, \text{ через } \psi_k \text{ (соответственно через } \bar{\psi}_k) \text{ — нумерацию, которую нумерация } \varphi \text{ инду-} \\ \text{цирует на } T(v_k) &\text{ (соответственно на } \bar{T}(v_k)). \text{ Очевидно,} \\ \Pi(T, \varphi) &= \Pi(\bar{T}(v_k), \bar{\psi}_k) + \Pi(T(v_k), \psi_k) + n - \psi_k(v_k). \quad (4.8) \end{aligned}$$

По лемме 2 и определению растянутости P_{ψ_k} получаем, что

$$\begin{aligned} \Pi(T(v_k), \psi_k) &\geq \Pi(T(v_k)) + P_{\psi_k} = \Pi(T(v_k)) + \\ &+ (\psi_k(v_k) - 1 - n_k + 1) = \Pi(T(v_k)) + \psi_k(v_k) - n_k. \quad (4.9) \end{aligned}$$

Так как число вершин в дереве $\bar{T}(v_k)$ с корнем v_0 меньше чем n , то по предположению индукции имеет место неравенство

$$\begin{aligned} \Pi(\bar{T}(v_k), \bar{\psi}_k) &\geq \\ &\geq \sum_{i=1, i \neq k}^d \Pi(T(v_i)) + n_2 + 2n_3 + \dots + (k-2)n_{k-1} + \\ &+ (k-1)n_{k+1} + kn_{k+2} + \dots + (d-2)n_d + (d-1). \end{aligned}$$

Подставим это неравенство вместе с (4.9) в (4.8):

$$\begin{aligned} \Pi(T, \varphi) &\geq \\ &\geq \sum_{i=1}^d \Pi(T(v_i)) + n_2 + 2n_3 + \dots + (k-2)n_{k-1} + \\ &+ (k-1)n_{k+1} + kn_{k+2} + \dots + (d-2)n_d + (d-1) + \\ &+ \sum_{i=1}^d n_i + 1 - n_k = \sum_{i=1}^d \Pi(T(v_i)) + n_1 + 2n_2 + 3n_3 + \dots \\ &\dots + (k-1)n_{k-1} + kn_{k+1} + (k+1)n_{k+2} + \dots \\ &\dots + (d-1)n_d + d. \end{aligned}$$

Вычитая из этого неравенства равенство (4.7), получаем, что

$$\begin{aligned} \Pi(T, \varphi) - \Pi(T, \varphi_0) &\geq (n_1 - n_2) + \\ &+ 2(n_2 - n_3) + \dots + (k-1)(n_{k-1} - n_k) \geq 0. \end{aligned}$$

Это означает, что нумерация φ_0 оптимальна.

По индукции, используя (4.7), нетрудно показать, что $\Pi(T) = \sum_{v \in V} b(v)$.

Чтобы построить нумерацию $A(T)$ дерева $T = (V, E)$, нужно определить количество вершин в каждом поддереве $T(v)$, $v \in V$, и упорядочить поддеревья $T(v_1), \dots, T(v_d)$, где v_1, \dots, v_d — потомки вершины v , по числу вершин в них. Можно подсчитать, что трудоемкость описанного способа построения нумерации $A(T)$ для n -вершинного дерева равна $O(n \log n)$.

1.5. Алгоритм построения оптимальной по длине укладки ордерова. Построение оптимальной по длине укладки сводится к заполнению таблицы с n строками по строке на каждую вершину, но уже с пятью столбцами. Пусть $\rho(i, j)$ обозначает содержимое клетки таблицы на пересечении i -й строки и j -го столбца.

Вход: Ордерова $T = (V, E)$ с корнем r , заданное списками смежности.

Выход: Оптимальная по длине укладка φ и длина ордерова $\Pi(T)$.

начало

1. провести поиск в ширину и занумеровать вершины в соответствии с порядком их обхода;

2. для i от n до 0 шаг -1 цикл

3. если номер вершины v равен i то $\rho(i, 1) := v$;

4. для i от 1 до n шаг 1 цикл

5. начало

6. если v из $\rho(i, 1)$ есть висющаяся вершина то

7. начало

8. $\rho(i, 2) := 1$;

9. $\rho(i, 3) := 0$;

10. $\rho(i, 4) := \rho(i-1, 4)$;

примечание для $i=1$ $\rho(1, 4) = 0$ конец примечания

11. $\rho(i, 5) := v$;

12. конец

13. иначе начало

14. упорядочить вершины y_1, \dots, y_d из Γv в порядке убывания содержимого $\rho(y_i, 2)$ второго столбца, пусть упорядоченная последовательность есть y_{i_1}, \dots, y_{i_d} ;

15. $\rho(i, 2) := 1; \rho(i, 3) := 1; \rho(i, 5) := \emptyset$;

16. для t от 1 до d шаг 1 цикл

17. начало

18. $\rho(i, 2) := \rho(i, 2) + \rho(y_{i_t}, 2)$;

19. $\rho(i, 3) := \rho(i, 3) + (t-1)\rho(y_{i_t}, 2)$;

20. $\rho(i, 5) := \rho(i, 5) \rho(y_{i_t}, 2)$;

21. конец цикла;

22. $\rho(i, 4) := \rho(i, 3) + \rho(i-1, 4)$;

23. $\rho(i, 5) := \rho(i, 5)v$;

24. конец

25. конец цикла;

26. $\Pi(T) := \rho(n, 4)$;

27. $\varphi_0 := \rho(n, 5)$;

конец

Итак, после заполнения таблицы в клетке 4 последней n -й строки (рис. 4.3) будет стоять длина дерева T , а в клетке 5 этой же строки будет выписана последовательность всех вершин этого дерева. Сопоставляя вершине, стоящей в этой последовательности на i -м

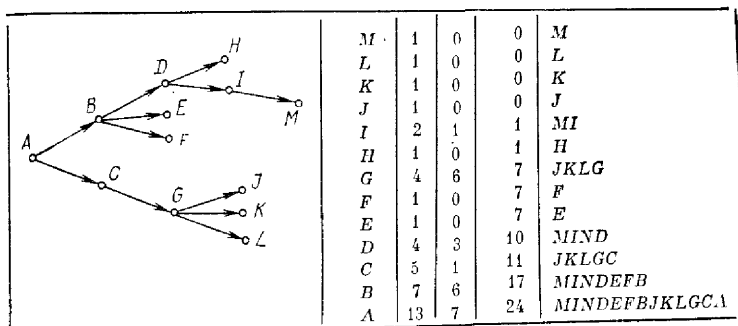


Рис. 4.3.

месте, номер i , получаем оптимальную нумерацию дерева. Пример использования алгоритма показан на рис. 4.3.

Приведем без доказательства оценки для длины и ширины ордеревьев. Пусть T^d есть n -вершинное дерево, полустепени исхода вершин которого не превосхо-

дят d ($d \geq 2$). Для таких деревьев имеют место следующие оценки:

— для длины

$$\begin{aligned} \frac{d-1}{4 \log d} n \log n + O(n \log n) &\leq \Pi(T^d) \leq \\ &\leq \frac{d-1}{2 \log d} n \log n + O(n \log n); \end{aligned}$$

— для ширины

$$W(T^d) = \frac{d-1}{\log d} \log n + O(\log n).$$

1.6. Укладки неориентированных деревьев. Точное решение задачи об оптимальной по длине укладке достаточно сложно. Приведем алгоритм построения плоских минимальных упадок с трудоемкостью $O(n)$, дающий приближенное решение, превосходящее соответствующую величину в классе всех упадок не более чем в полтора раза.

Выделим класс упадок Φ^* , каждой укладке которого соответствует разложение дерева на последовательность цепей σ_j , $j \in [1:l]$, обладающих следующими свойствами:

- а) концевые вершины цепей являются висячими вершинами в поддеревьях, в которых они выделяются;
- б) нумерация вершин каждой цепи монотонна относительно возрастания или убывания номеров, а нумерующие последовательности всех поддеревьев, образующихся в процессе разложения на цепи, сплошные.

Можно показать, что все минимальные укладки содержатся в классе Φ^* . Нетрудно увидеть, что концевые вершины цепей σ_j , $j \in [1:l]$, — вершины нечетной степени, а $l = n$ при $2n$ вершинах нечетной степени в произвольном дереве.

Хотя класс плоских нумераций $\Phi_{пл}$ не совпадает с классом Φ^* и ни один из них не содержится целиком в другом, справедлива

Лемма 3. *Всякая минимальная плоская укладка принадлежит классу Φ^* .*

Доказательство этой и следующей леммы мы опускаем.

В силу этой леммы в дальнейшем имеет смысл рассматривать только те плоские укладки, которые входят в класс Φ^* .

Лемма 4. Укладка φ_k из класса Φ^* является плоской тогда и только тогда, когда подграф $\bigcup_{j=1}^k \sigma_j$ связан при любом $k \in [2 : l-1]$.

Установим необходимое и достаточное условие минимальности плоских упаковок. Выделим в дереве $T = (V, E)$ произвольную вершину v_i степени $p(v_i)$ и все инцидентные ей ребра (v_i, v_{j_r}) , $r \in [1 : p]$. Любая вершина v_m , $m \neq i$, связана с вершиной v_i единственной цепью. Будем говорить, что вершина v_m принадлежит ветке, выходящей из v_i по ребру (v_i, v_{j_r}) , $r \in [1 : p]$, если в вершину v_m можно попасть из v_i по цепи, содержащей ребро (v_i, v_{j_r}) . Число всех таких вершин (число вершин в ветке) обозначим через n_{j_r} , $r \in [1 : p]$.

Рассмотрим произвольную укладку $\varphi \in \Phi^*$. Упорядочим ветки к каждой вершине v_i в соответствии с порядком прохождения через v_i цепей из $\{\sigma_j\}$, $j \in [1 : l]$. Будем называть полученный порядок веток n_i -допустимым, если не найдется ни одной пары веток к произвольной вершине v_i степени $\deg(v_i) \geq 3$ с числом вершин n_r и n_q , $r, q \in [1 : p]$, таких, что

$$n_r < n_q, \text{ где } r < \begin{cases} q & \text{при четном } r; \\ q-1 & \text{при нечетном } r. \end{cases}$$

Способ выделения цепей σ_j , $j \in [1 : l]$, индуцирующий n_i -допустимый порядок веток, также будем называть n_i -допустимым.

Теорема 5. Плоская укладка $\varphi \in \Phi^*$ произвольного дерева $T = (V, E)$ минимальна тогда и только тогда, когда способ выделения цепей σ_j , $j \in [1 : l]$, является n_i -допустимым.

Доказательство. Необходимость. Пусть φ — минимальная плоская укладка. Для вершин степени $\deg(v_i) \leq 2$ необходимость условия теоремы очевидна. Рассмотрим произвольную вершину v_i степени $\deg(v_i) \geq 3$. Возможны два случая.

1. Цепь σ_1 проходит через вершину v_i . При этом из лемм 3 и 4, а также из свойств упаковок класса Φ^* следует, что нумерующие последовательности всех веток к вершине v_i будут сплошными. Взаиморасположение их вдоль числовой прямой приведено на рис. 4.4, где n_{2s-1} и n_{2s} , $s \in [1 : \lfloor p/2 \rfloor]$, суть числа вер-

шин в ветках, по которым s -я цепь разложения проходит через v_i . При нечетной степени $\deg(v_i)$ указаны две возможности для нумерации вершины v_i , поскольку здесь последняя цепь разложения проходит через v_i только по одной ветке (с числом вершин n_p) и в зависимости от выбора направления нумерации на этой цепи последняя ветка может быть занумерована как до, так и после вершины v_i .

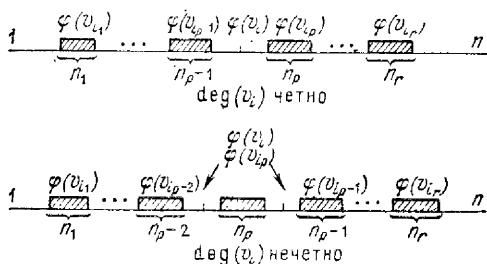


Рис. 4.4.

Предположим, что условие теоремы не выполнено для вершины v_i , т. е. найдется хотя бы одна пара веток к вершине v_i , порядок выделения которых не является n_r -допустимым. Построим тогда укладку $\varphi' \in \Phi^*$, отличающуюся от φ лишь порядком нумерации этих веток (пусть число вершин в них равно n_r и n_q , $r < q$). Нумерующие последовательности всех веток к вершине v_i сохраним сплошными, оставив прежним и порядок нумерации внутри каждой ветки. При этом длины ребер (длина ребра есть модуль разности номеров его концов) могут измениться лишь у ребер, инцидентных v_i . Рассмотрим характер соответствующего изменения длины всего дерева. При нечетном r имеем

$$\sum_{j=1}^{r-1} |\varphi(v_i) - \varphi(v_{i_j})| = \sum_{j=1}^{r-1} |\varphi'(v_i) - \varphi'(v_{i_j})|,$$

так как $\varphi'(v_{i_j}) = \varphi(v_{i_j})$ для $j \leq r-1$, а число вершин v_{i_j} с $\varphi(v_{i_j}) < \varphi(v_i)$ ($\varphi'(v_{i_j}) < \varphi'(v_i)$) равно числу вершин v_{i_j} с $\varphi(v_{i_j}) > \varphi(v_i)$ ($\varphi'(v_{i_j}) > \varphi'(v_i)$). При четном r по таким же соображениям приходим к равенству

$$\sum_{j=1}^{r-2} |\varphi(v_i) - \varphi(v_{i_j})| = \sum_{j=1}^{r-2} |\varphi'(v_i) - \varphi'(v_{i_j})|.$$

Величина $\sum_{j=r}^p |\varphi(v_i) - \varphi(v_{ij})|$ для нечетного r или же величина $\sum_{j=r-1}^p |\varphi(v_i) - \varphi(v_{ij})|$ для четного r обязательно уменьшится при переходе к укладке φ' , и не менее чем на $n_q - n_r$.

Итак, длина укладки $\Pi(T, \varphi) = \sum_{(v_i, v_j)} |\varphi(v_i) - \varphi(v_j)|$ уменьшается при переходе от укладки φ к укладке φ' . Нетрудно увидеть, что φ' — плоская укладка, если таковой является φ . Таким образом, мы пришли к выводу, что φ не является минимальной плоской укладкой, как это утверждалось. Следовательно, предположение о невыполнении условия теоремы для произвольной вершины v_i , лежащей на цепи σ_i , неверно. Отсюда, в частности, следует, что при минимальной плоской нумерации цепь σ_i проходит через вершины дерева по веткам с наибольшим числом вершин.

2. Цепь σ_i не содержит вершину v_i . Пусть $\sigma_k, k \in [2: l-1]$, — первая цепь разложения, проходящая через v_i . Так как $\bigcup_{j=1}^k \sigma_j$ связно (по лемме 4), то цепь σ_k обязательно проходит через v_i по ветке, содержащей предыдущие выделенные цепи $\sigma_j, j \in [1: k-1]$. Число вершин в той ветке больше, чем в какой-либо другой ветке к v_i , так как она содержит цепь σ_i , проходящую через вершины дерева по веткам с наибольшим числом вершин. Следовательно, нарушение условия n_i -допустимости невозможно ни на одной паре веток к вершине v_i , содержащей ветку с σ_i . Поскольку нумерация остальных веток к вершине v_i устроена так же, как и в части 1 доказательства (сплошные нумерующие последовательности), то дальнейшее доказательство проводится аналогично.

Достаточность. Нетрудно убедиться в том, что если способ выделения цепей $\sigma_j, j \in [1: l]$, удовлетворяет условию теоремы, то $\bigcup_{j=1}^h \sigma_j$ связно при всех $h \in [2: l-1]$; любая укладка $\varphi \in \Phi^*$ является при этом плоской (лемма 4). Все такие укладки обладают, кроме того, одинаковой длиной. Действительно, если укладки отличаются лишь направлением роста номеров вдоль некоторых цепей из $\{\sigma_j\}, j \in [1: l]$, то справедливость доказываемого утверждения непосредствен-

но следует из свойств укладок класса Φ^* и определения длины укладки. Если меняется разложение на цепи σ_j , $j \in [1: l]$, то это влияет лишь на порядок нумерации веток с одинаковым числом вершин для некоторых вершин v_i . Из способа выделения цепей следует, что ветки к любой вершине v_i , содержащие одинаковое число вершин, имеют сплошные нумерующие последовательности. Поэтому, меняя порядок нумерации таких веток, всегда можно перейти от одного разложения к другому, сохранив неизменной длину укладки. Отсюда, поскольку минимальные плоские нумерации содержатся среди рассматриваемых нумераций (это следует из необходимости условия теоремы), получаем минимальность всех укладок, удовлетворяющих условию теоремы.

Замечание. Теорему 5 удобно сформулировать для дальнейшего изложения так:

Теорема 6. *Плоская укладка произвольного дерева является минимальной тогда и только тогда, когда цепи σ_j , $j \in [1: l]$, проходят через каждую вершину по еще не пройденным веткам с наибольшим числом вершин.*

Основываясь на теореме 5, можно предложить эффективный алгоритм построения минимальной плоской укладки произвольного дерева.

1.7. Алгоритм построения минимальной плоской укладки. Пусть дерево $T = (V, E)$ с n вершинами задано списком ребер.

Алгоритм:

Шаг 1. Превратить исходное дерево в корневое ордерное с корнем в центроиде дерева.

Шаг 2. Упорядочить массивы $\Gamma^{-1}v_i$, $i \in [1: n]$, по весам вершин.

Шаг 3. Выделить цепи σ_j , $j \in [1: l]$.

Шаг 4. Построить минимальную укладку вершин дерева.

Раскроем содержание каждого шага.

Шаг 1. Для выполнения этого шага необходимо провести ранжирование вершин, начиная с висячих, а также сформировать множества $\Gamma^{-1}v_i$ и Γv_i . При ранжировании вершин каждая из них рассматривается в качестве корня поддерева, состоящего из вершин меньшего ранга, и ей присваивается вес, равный числу вершин в соответствующем поддереве. Первая вер-

шина v_i , вес которой равен $n_i \geq n/2$ (она, очевидно, является центроидальной), рассматривается в качестве корня всего дерева, ей присваивается максимальный ранг после окончания процесса ранжирования всех остальных вершин дерева, а вес n_i полагается равным n . Этот шаг можно выполнить за $O(n)$ операций.

Шаг 2. За один просмотр массивов $\Gamma^{-1}v_i$, $i \in [1:n]$, формируем массив, в котором в ячейке с адресом $x = x_0 + n_i$ будет содержаться число вершин с весом, равным n_i . Теперь нетрудно найти адреса, начиная с которых будут расположены массивы $V(n_i)$ вершин веса n_i . За второй просмотр массивов $\Gamma^{-1}v_i$, $i \in [1:n]$, рассортируем входящие в них вершины по массивам $V(n_i)$, последовательно заполняя каждый из них. Заново сформируем массивы $\Gamma^{-1}v_i$, $i \in [1:n]$. Для этого последовательно просматриваем все массивы $V(n_i)$ и, используя массивы Γv_i , определяем для очередной вершины v_i , $i \in [1:n]$, ее потомка v_j , $j \in [1:n]$, $j \neq i$, после чего заносим вершину в очередную ячейку массива $\Gamma^{-1}v_j$. Очевидно, что после просмотра всех вершин из V мы получим массивы $\Gamma^{-1}v_i$, каждый из которых упорядочен в порядке возрастания весов входящих в них вершин.

Этот шаг можно выполнить за $O(n)$ операций.

Шаг 3. Поскольку $\bigcup_{j=1}^k \sigma_j$ связно при любом k , $k \in [2:l-1]$, то в каждом поддереве разложения можно заранее указать вершину (опорную), через которую пройдет первая цепь его разложения (опорной вершиной всего дерева является центроид дерева). Поддеревья разложения определяются следующим образом. Пусть $T = (V, E)$ — дерево с n вершинами, на котором задана некоторая нумерация φ . Выделим в дереве цепь $\sigma_i = (V_i, E_i)$, соединяющую вершины $\varphi^{-1}(1)$ и $\varphi^{-1}(n)$. Каждой вершине $v_i \in V_i$, степень которой больше или равна трем, поставим в соответствие поддерево $T_i^{\sigma_i} = (V_i^1, E_i^1)$, $i \in [1:k_1]$, множество вершин которого состоит из v_i и всех вершин, которые соединяются с вершиной v_i цепями, не содержащими ребер из E_i . Множество E_i^1 образовано ребрами этих цепей. Будем называть поддеревья $T_i^{\sigma_i} = (V_i^1, E_i^1)$ *поддеревьями разложения*. В каждом таком поддереве снова выделим цепь, соединяющую вершины с наибольшим и наи-

меньшим номерами, что приведет к образованию новых поддеревьев разложения. Этот процесс продолжается до тех пор, пока не будет получено разложение дерева на пореберно непересекающиеся цепи.

Используя опорные вершины в качестве исходных, а также упорядоченные массивы $\Gamma^{-1}v_i$, $i \in [1:n]$, производим выделение цепей σ_j , $j \in [1:l]$, проходящих через вершины v_i , $i \in [1:n]$, по не пройденным веткам с наибольшим числом вершин. При этом производится модификация веса каждой вершины v_i . При модификации текущее (начальное) значение веса вершины v_i уменьшается на величину веса той вершины из $\Gamma^{-1}v_i$, по которой проходит очередная цепь σ_j , $j = 1, \dots, l$.

Шаг 4. Пусть $\{v_i^j\}$, $i \in [1:k_j]$, есть множество вершин цепи σ_j , $j \in [1:l]$. Обозначим через $\varphi_{\min}(v_q^j)$, $j \in [1:l]$, $q \in [1:k_j]$, наименьший номер, который может иметь вершина v_q^j при заданном направлении возрастания номеров вершин вдоль цепей σ_m , $m \in [1:j]$. Так как мы строим укладку φ из класса Φ^* , то для упрощения обозначений всегда оказывается возможным считать, что индекс i перечисляет вершины вдоль каждой цепи σ_j в порядке возрастания номеров. Обозначим модифицированный вес вершины v_i^j через n_i^j . При этом получаем для $\varphi_{\min}(v_q^j)$ следующее выражение:

$$\varphi_{\min}(v_q^j) = \varphi(v_1^j) + \sum_{i=1}^{q-1} n_i^j, \quad j = 1, \dots, l, \quad q = 1, \dots, k_j,$$

причем $\varphi(v_1^1) = 1$, $\varphi(v_i^j) = \varphi_{\min}(v_r^m)$, где v_r^m , $m \in [1:j-1]$, $r \in [1:k_m]$, — опорная вершина поддерева, в котором выделяется цепь σ_j , $j \in [2:l]$. Для вершин v_i^j , у которых на данном этапе разложения веса равны $n_i^j = 1$, имеем $\varphi_{\min}(v_i^j) = \varphi(v_i^j)$, т. е. каждая вершина получает свой окончательный номер при выделении последней проходящей через нее цепи σ_j . Очевидно, что число операций, необходимых для расстановки указанным способом номеров вершин дерева, не превосходит $O(n)$. Отсюда следует, что трудоемкость всего алгоритма равна $O(n)$. Расход памяти равен $O(n \log n)$.

Приведем без доказательства оценки для длины неориентированных деревьев. Имеем:

— для класса всех n -вершинных деревьев

$$\Pi = \lfloor n^2/4 \rfloor;$$

— для класса n -вершинных деревьев, степени вершин которых не превосходят $d \geq 3$,

$$\frac{d-7/3}{12 \log d} n \log n \left(1 - \frac{\log(3/2)}{\log n} \right) < \Pi < \frac{d+4}{4 \log(d-1)} n \log n.$$

1.8. Укладки произвольных графов. Известны следующие оценки.

Для ширины графов:

— для класса n -вершинных бесконтурных орграфов

$$W = \lfloor n/2 \rfloor;$$

— для класса n -вершинных бесконтурных орграфов, степени вершин которых не превосходят d ,

$$d'' \log n \leq W \leq d \log n;$$

— для класса произвольных орграфов, степени вершин которых не превосходят d ,

$$\alpha_0(n-1) \leq W \leq dn/2$$

(здесь $\alpha_0 \in (0, \alpha_1)$, а α_1 — такая положительная константа, что $\alpha_1 n < (3/2)p'n - 1$, p' — число, определяемое соотношением $\lfloor pn \rfloor = p'n$, $p \in (0, 1/2]$).

Для длины графов:

— для класса D бесконтурных орграфов

$$\gamma n^2 \leq \Pi(D) \leq A n^2,$$

где γ, A — коэффициенты, не зависящие от n ;

— для класса графов, степени вершин которых не превосходят d ,

$$\gamma \alpha_0 n^2 \leq \Pi(\Gamma^d) \leq (1/2) A d n^2,$$

где α_0 определяется, как и выше, а γ и A — коэффициенты, не зависящие от n ;

— для класса D^d бесконтурных орграфов, степени вершин которых не превосходят d ,

$$d' n \log n \leq \Pi(D^d) \leq d n \log n.$$

Пусть $W(G, \varphi, t)$ — ширина укладки φ n -вершинного графа G в вершине t .

Теорема 7. Если φ — укладка n -вершинного графа G , то

$$\Pi(G, \varphi) = \sum_{i=1}^{n-1} W(G, \varphi_i, t).$$

Следствие 1. Если φ_0 — оптимальная по ширине укладка, а $\Pi(G) = \min_{\varphi} \Pi(G, \varphi)$, где минимум берется по всевозможным укладкам, то

$$\Pi(G) \leq \Pi(G, \varphi_0) =$$

$$= \sum_{t=1}^{n-1} W(G, \varphi, t) \leq (n-1) W(G, \varphi_0) = (n-1) W(G).$$

Следствие 2. Если G получается из графа G добавлением k вершин степени два на ребра графа G , то $W(G^{(k)}) = W(G)$ и $\Pi(G^{(k)}) \leq (n+k-1)W(G)$.

1.9. Построение оптимальных по числу соседней связности упадок. Пусть φ — допустимая укладка орграфа $G = (V, E)$. Будем говорить, что вершины v_k и v_{k+1} находятся в отношении соседней связности, если $(v_{k+1}, v_k) \in E$. Числом соседней связности укладки φ называется величина

$$\kappa(\varphi) = \sum_{k=1}^{n-1} \delta(v_{k+1}, v_k),$$

где $\delta(v_{k+1}, v_k) = 1$, если $(v_{k+1}, v_k) \in E$, и 0 в противном случае.

Компонентой укладки φ назовем последовательность вершин этой укладки $C = (v_{j_1}, \dots, v_{j_t})$, удовлетворяющих условиям:

- а) $\delta(v_{j_1}, v_{j_{t-1}}) = 0$;
- б) $\delta(v_{j_1+1}, v_{j_t}) = 0$;
- в) $\delta(v_{j_p+1}, v_{j_p}) = 1, \quad p = 1, \dots, t-1$,

Очевидно, что всякую укладку φ можно представить в виде последовательности компонент $\varphi = (C_1, \dots, C_q)$. Будем называть C_i компонентой i -го ранга.

Лемма 5. Если q — число компонент в укладке φ , то $\kappa(\varphi) = n - q$.

Числом соседней связности графа G называется величина $\kappa(G) = \max_{\varphi \in \Phi(G)} \{\kappa(\varphi)\}$, где $\Phi(G)$ — множество всех упадок графа G . Укладка $\varphi_0 \in \Phi(G)$ называется оптимальной по числу соседней связности, если $\kappa(\varphi_0) = \kappa(G)$.

Точное решение задачи построения оптимальной укладки имеет экспоненциальную трудоемкость, поэтому для решения практических задач целесообразно использовать эвристические алгоритмы. Введем вспомогательные определения.

Путь $\mu = [x_{i_1}, \dots, x_{i_p}]$ в графе G называется *проникающим*, если

а) $\deg^+(x_{i_1}) = 0$;

б) в графе G отсутствуют дуги вида (y, x_{i_q}) , где $q \in [1: p]$ и $y \in V \setminus \bigcup_{q=1}^p \{x_{i_q}\}$.

Проникающий путь $\mu = [x_{i_1}, \dots, x_{i_p}]$ называется *максимальным*, если в графе G не найдется вершины y такой, что путь $\mu' = [x_{i_1}, \dots, x_{i_p}, y]$ также является проникающим.

Назовем дугу (x, y) *избыточной*, если в графе G существует хотя бы один путь μ из x в y длины $l(\mu) \geq 2$. Можно показать, что удаление избыточных дуг не изменяет числа соседней связности. Пусть граф G' получается из графа G удалением всех избыточных дуг; при этом всякой укладке $\varphi(G)$ графа G однозначно соответствует укладка $\varphi'(G)$ графа G' .

Алгоритм:

Вход: Орграф $G = (V, E)$, заданный списками смежности.

Выход: Укладка φ , являющаяся приближенным решением задачи.

начало

1. удалить все избыточные дуги;

 примечание теперь граф G превратился в G'

конец примечания

2. $G_1 := G'$;

3. для i от 1 пока $G_i \neq \emptyset$ цикл

4. **начало**

5. в графе G_i определить множество M_i проникающих путей;

6. выбрать в M_i путь μ_i^* наибольшей длины;

7. $C_i := \mu_i^*$;

8. $G_{i+1} := G_i \setminus \mu_i^*$;

 примечание при удалении пути μ_i^* удаляются все его вершины и инцидентные им дуги **конец примечания**

9. **конец цикла**;

10. $\varphi := (C_1, C_2, \dots)$;

конец

Для определения максимального проникающего пути на шаге 6 можно воспользоваться методом наход-

дения критического пути в бесконтурном орграфе. Трудоемкость алгоритма равна $O(n^2)$. Повысить точность получаемого решения можно за счет введения операции прогноза на один шаг.

Алгоритм с прогнозом:

Вход: Орграф $G = (V, E)$, заданный списками смежности.

Выход: Укладка $\varphi = (C_1, \dots, C_k)$, являющаяся приближенным решением задачи.

начало

1. удалить все избыточные дуги;

2. $G_1 := G'$;

3. для i от 1 пока $G_i \neq \emptyset$ цикл

4. начало

5. в G_i определить множество M_i проникающих путей;

6. для каждого пути $\mu \in M_i$ цикл

7. начало

8. $G_i(\mu) := G_i \setminus \mu$;

9. в $G_i(\mu)$ найти максимальный проникающий путь $\pi_i^*(\mu)$;

10. конец цикла;

11. среди всех путей $\mu \in M_i$ выбрать путь μ_i^* , для которого величина $l(\mu_i^*) + l(\pi_i^*(\mu))$ наибольшая;

12. $C_i := \mu_i^*$;

13. $G_{i+1} := G_i \setminus \mu_i^*$;

14. конец цикла;

15. $\varphi := (C_1, C_2, \dots)$;

конец

§ 2. Конструирование оптимальных деревьев

В параграфе будут рассмотрены различные критерии оптимальности корневых деревьев и каркасов неориентированных графов, а также алгоритмы построения оптимальных и почти оптимальных деревьев.

2.1. Оптимальные бинарные деревья. *Бинарное дерево* определяется рекурсивно следующим образом:

— пустое дерево, состоящее из единственной вершины v_0 , есть бинарное дерево;

— бинарное дерево T_n с $n \geq 1$ вершинами есть упорядоченная тройка (T_l, v_0, T_r) , где v_0 — корень дерева T_n , а T_l и T_r — бинарные деревья с n_l и n_r верши-

нами соответственно, называемые *левым* и *правым* бинарными деревьями ($n_l \geq 0$, $n_r \geq 0$, $n_l + n_r = n - 1$).

Бинарное дерево представляет собой ориентированное дерево, у которого в каждую вершину, отличную от корня, входит только одна дуга, а выходит не более двух. При этом для каждой выходящей дуги известно, является ли она правой или левой. Каждая вершина бинарного дерева, отличная от корня, может рассматриваться как корень бинарного поддерева с вершинами, достижимыми из нее. Бинарное поддерево с корнем в вершине x будет обозначаться через $T(x)$ (рис. 4.5).

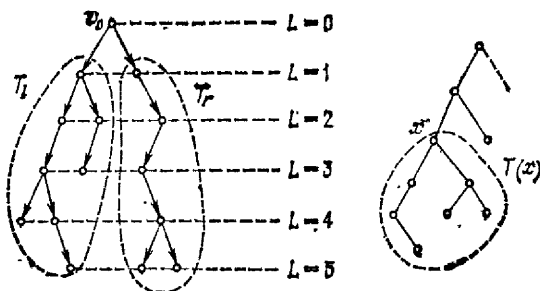


Рис. 4.5.

Уровнем $L(x)$ вершины x в n -вершинном дереве T_n называется длина пути из v_0 в x . Корень v_0 есть единственная вершина уровня 0. При графическом изображении дерева вершины одного уровня располагаются на одной прямой.

Взвешенным корневым деревом \mathcal{T}_n над последовательностью неотрицательных весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$ называется бинарное дерево T_{2n+1} с n внутренними вершинами v_1, v_2, \dots, v_n , каждой из которых сопоставлен вес β_i , и с $n+1$ висячими вершинами w_2, w_1, \dots, w_n , каждой из которых сопоставлен вес α_j .

Пример взвешенного корневого дерева над последовательностью весов $[4, 2, 6, 1, 1, 1, 8, 7, 5, 8, 4, 2, 1]$ показан на рис. 4.6.

Стоимостью взвешенного корневого дерева \mathcal{T}_n над последовательностью весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$ называется величина

$$C(\mathcal{T}_n) = \sum_{i=1}^n \beta_i (L(v_i) + 1) + \sum_{j=0}^n \alpha_j L(w_j).$$

Для дерева на рис. 4.6 стоимость равна 131.

2.2. Построение оптимальных деревьев. Алгоритм основан на следующем принципе оптимальности: оптимальное бинарное дерево T_n над последовательностью весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$ имеет корнем вершину с весом β_i , оптимальное бинарное дерево T_{i-1} над последовательностью весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_{i-1}, \alpha_{i-1}$ в качестве левого поддерева и оптимальное дерево T_{n-i} над последовательностью весов $\alpha_i, \beta_{i+1}, \alpha_{i+1}, \dots, \beta_n, \alpha_n$ в качестве правого поддерева.

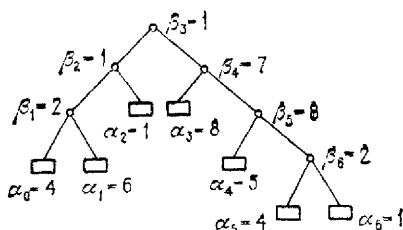


Рис. 4.6.

Пусть $C(i, j)$ — стоимость оптимального поддерева над последовательностью весов $\alpha_i, \beta_{i+1}, \alpha_{i+1}, \dots, \beta_j, \alpha_j$, и пусть

$$w(i, j) = \beta_{i+1} + \dots + \beta_j + \alpha_i + \dots + \alpha_j, \quad 0 \leq i \leq j \leq n,$$

есть сумма этих весов. Так как минимально возможная стоимость дерева с корнем в вершине с весом β_k равна $w(i, j) + C(i, k-1) + C(k, j)$, то имеем рекуррентное соотношение, вытекающее из сформулированного принципа оптимальности:

$$C(i, j) = 0,$$

$$C(i, j) = w(i, j) + \min_{i \leq k \leq j} (C(i, k-1) + C(k, j)), \quad i < j. \quad (*)$$

Если $i < j$, то пусть $R(i, j)$ обозначает множество всех индексов, при которых достигается минимум в (*); это множество определяет возможные корни оптимальных деревьев. Как показал Д. Кнут, нет нужды искать все множество $R(i, j)$; достаточно найти только те элементы $r(i, j)$ этого множества, которые удовлетворяют неравенству

$$r(i, j-1) \leq r(i, j) \leq r(i+1, j)$$

при условии неотрицательности весов. Теперь в (*) вместо $j-i$ значений k нужно проверять только

$r(i+1, j) - r(i, j-1) + 1$ значений. Это позволяет сократить трудоемкость до $O(n^2)$ при требуемой памяти $O(n^2)$.

Алгоритм К:

Пусть даны неотрицательные веса $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$. Алгоритм К последовательно строит оптимальные бинарные деревья $t(i, j)$ для весов $\alpha_i, \beta_{i+1}, \dots, \beta_j, \alpha_j$. Для этого вычисляются величины: $C(i, j)$ — стоимость дерева $t(i, j)$, $0 \leq i \leq j \leq n$; $r(i, j)$ — корень дерева $t(i, j)$, $0 \leq i \leq j \leq n$; $w(i, j)$ — полный вес дерева $t(i, j)$, $0 \leq i \leq j \leq n$. Результаты работы алгоритма определяются величинами $r(i, j)$: если $i = j$, то дерево $t(i, j)$ пусто; если $i \neq j$, то левым поддеревом является дерево $t(i, r(i, j) - 1)$, а правым — $t(r(i, j), j)$.

Вход: Последовательность весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$.

Выход: Оптимальное дерево над данной последовательностью весов.

начало

1. для i от 0 до n шаг 1 цикл
2. **начало**
3. $C(i, i) = 0$;
4. $w(i, i) := \alpha_i$;
5. для j от $i+1$ до n шаг 1 цикл
6. **начало**
7. $w(i, j) := w(i, j-1) + \beta_j + \alpha_j$;
8. **конец цикла**;
9. **конец цикла**;
10. для j от 1 до n шаг 1 цикл
11. **начало**
12. $C(j-1, j) := w(j-1, j)$;
13. $r(j-1, j) := j$;
14. **конец цикла**;
15. для d от 2 до n шаг 1 цикл
16. **начало**
17. для j от d до n шаг 1 цикл
18. **начало**
19. положить $i := j - d$;
20. положить k равным тому индексу k , при котором выражение $C(i, k-1) + C(k, j)$ достигает минимума;
21. $C := \min (C(i, k-1) + C(k, j))$;
22. $C(i, j) := w(i, j) + C$;
23. **конец цикла**;

24. конец цикла;

примечание после завершения цикла по d будут определены параметры оптимального дерева и всех его оптимальных поддеревьев **конец примечания**
конец

Применение алгоритма K на практике нецелесообразно, ибо можно построить почти линейными алгоритмами деревья со стоимостью, лишь незначительно отличающейся от оптимальной.

2.3. Построение почти оптимальных деревьев. Рассмотрим некоторые элементарные преобразования деревьев. Первое преобразование определяется так. Пусть $\{T^{(n)}\}$ — множество n -вершинных бинарных деревьев, $T_1 \in \{T^{(n)}\}$, v — некоторая вершина из T_1 . Тогда:

а) если v — корень дерева T_1 , то $\varphi(v, T_1) = T_1$;

б) если v — корень левого поддерева вершины w , то $\varphi(v, T_1) = T_2$, где T_2 получается из T_1 поднятием вершины v на место вершины w , так что w становится корнем правого поддерева вершины v (при этом вершина v и каждая вершина из $T_l(v)$ поднимается вверх на один уровень, а вершина w и каждая вершина из $T_r(w)$ опускается на один уровень);

в) если v — корень правого поддерева, то преобразование $\varphi(v, T_1)$ определяется симметричным способом.

Преобразование $\varphi(v, T)$ показано на рис. 4.7.

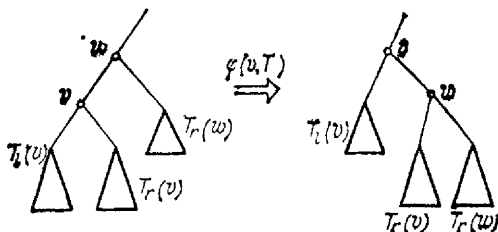


Рис. 4.7.

Более общее преобразование $\varphi^{(k)}(v, T)$ состоит в поднятии вершины v на k уровней. Оно заключается в k -кратном применении преобразования $\varphi(v, T)$, т. е. $\varphi^{(1)}(v, T) = \varphi(v, T)$ и $\varphi^{(k)}(v, T) = \varphi(v, \varphi^{(k-1)}(v, T))$. Преобразование $\varphi^{(k)}(v, T)$ показано на рис. 4.8.

На использовании преобразований $\varphi(v, T)$ и $\varphi^{(k)}(v, T)$ основан алгоритм построения почти оптимальных деревьев.

Вход: Последовательность весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$, параметр k .

Выход: Почти оптимальное дерево T .

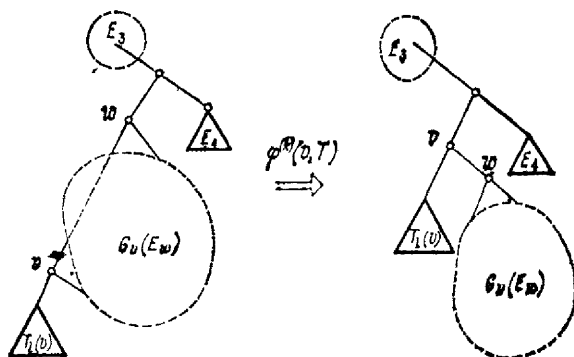


Рис. 4.8.

начало

1. выбрать в качестве начального приближения какое-нибудь бинарное дерево T над последовательностью весов $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$, например дерево, показанное на рис. 4.9;

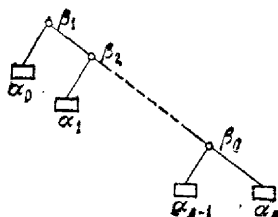


Рис. 4.9.

2. вычислить стоимость $c(T)$ дерева T ;
3. для всех $v \in V(T)$ цикл
4. начало
5. $T(v) := \varphi(v, T)$;
6. вычислить стоимость $c(T(v))$ дерева $T(v)$;
7. если $c(T(v)) < c(T)$ то
8. начало
9. $T := T(v)$;
10. $c(T) := c(T(v))$;
11. $f := \text{истина}$;
12. конец;

примечание логическая переменная f фиксирует факт существования хотя бы одного дерева $T(v)$, стоимость которого меньше стоимости дерева T **конец примечания**

13. если f то переход к шагу 12;

14. иначе стоп;

конец

Данный алгоритм осуществляет локальный поиск по множеству деревьев $\{T^{(n)}\}$. Точность решения и скорость работы алгоритма существенно зависят от величины параметра k .

Развитием данного алгоритма является top-down-алгоритм. Пусть $\bar{W}(i, j)$ обозначает множество весов $\{\beta_m | i \leq m \leq j\}$, а $\bar{T}(i, j)$ — множество бинарных деревьев на этом множестве весов. Для дерева $T(i, j)$ из $\bar{T}(i, j)$ в качестве корня берется вершина, соответствующая весу β_m , причем β_m удовлетворяет условиям:

а) β_m велико;

б) разность между стоимостями поддеревьев $T(i, m-1)$ и $T(m+1, j)$ мала;

в) разность между высотами (наибольшими значениями уровня) поддеревьев $T(i, m-1)$ и $T(m+1, j)$ мала.

Для интерпретации условий примем в качестве меры баланса стоимостей правого и левого поддеревьев дерева $T(i, j)$ величину

$$\sigma_m = \min \left\{ \frac{w_l(m) + \beta_m}{w + \beta_m}, \frac{w_r(m) + \beta_m}{w + \beta_m} \right\},$$

$$\text{где } w_l(m) = \sum_{s=i}^{m-1} \beta_s, \quad w_r(m) = \sum_{s=m+1}^j \beta_s, \quad w = \sum_{s=i}^j \beta_s.$$

В качестве меры баланса высот тех же поддеревьев примем величину

$$\tau_m = \min \left\{ \frac{1 + d_l(m)}{d}, \frac{1 + d_r(m)}{d} \right\},$$

где $d_l(m) = \log(m - i + 1)$, $d_r(m) = \log(j - m + 1)$, $d = \log(j - i + 2)$.

Top-down-алгоритм $A(i, j)$ представляет собой рекурсивную процедуру конструирования оптимального дерева $T^*(i, j)$ для множества весов $\bar{W}(i, j)$. Пусть $f(n)$ — функция, выбор которой определяет степень

приближения решения, и оптимальному. Функция может иметь вид:

а) $f(n) = M$, где M — константа;

б) $f(n) = F(n)$, где $F(n)$ — монотонно возрастающая функция, $1 \leq F(n) \leq n$;

в) $f(n) = \max(F(n), M)$.

Алгоритм $A(i, j)$:

Вход: Множество весов β_i, \dots, β_j (все $\alpha_k, k \in [0 : n]$, полагаем равными нулю).

Выход: Почти оптимальное дерево $T^*(i, j)$.

начало

1. процедура ДЕРЕВО(i, j);

2. начало

3. для k от i до j шаг 1 цикл

4. начало

5. вычислить $H(k)$;

примечание $H(k)$ есть одна из функций $H_1(k) = \beta_k \times \sigma_k$ или $H_2(k) = \beta_k \times \sigma_k + \tau_k$ конец примечания

6. конец цикла;

7. сформировать множество INT из $f(n)$ весов из $\bar{W}(i, j)$, которые имеют наибольшие значения функции $H(k)$;

8. построить алгоритмом K оптимальное дерево над множеством весов INT , налагая еще одно ограничение: каждая вершина либо не должна иметь потомков, либо иметь их ровно два;

9. $w_{k^*} :=$ корень построенного оптимального дерева;

10. ДЕРЕВО($i, k^* - 1$);

11. ДЕРЕВО($k^* + 1, j$);

12. сформировать оптимальное дерево для $\bar{W}(i, j)$ в виде $(T^*(i, k^* - 1), w_{k^*}, T^*(k^* + 1, j))$;

13. конец процедуры ДЕРЕВО;

14. $n := j - i + 1$;

15. если $n = 1$ то взять в качестве оптимального дерева дерево вида $T^*(i, j) = (\emptyset, w_i, \emptyset)$;

16. если $n \geq 2 \wedge n \leq f(n)$ то построить оптимальное дерево над множеством весов $\bar{W}(i, j)$ алгоритмом K ;

17. если $n > f(n)$ то ДЕРЕВО(i, j);

конец

Трудоёмкость алгоритма равна $O(n^{3/2} \log n)$, если $f(n) = M$, и $O(n^{3/2})$ в остальных случаях. Трудоёмкость может быть понижена за счет использования более тонкого выбора множества INT .

2.4. Статистический анализ почти оптимальных деревьев. Для построения почти оптимальных деревьев чаще всего используются две эвристики, которые мы сформулируем в виде правил.

Правило 1. Размещать вершины с большим весом как можно ближе к корню; аналогично поступать с каждым поддеревом.

Правило 2. Выбирать корень так, чтобы уравновесить полные веса правого и левого поддеревьев; аналогично поступать с каждым поддеревом.

Упомянутый во втором правиле вес поддерева может измеряться по-разному. С двумя подходами к измерению весов мы ознакомились при описании алгоритма $A(t, f)$, третий состоит в принятии в качестве веса поддерева числа его вершин.

Для оценки этих и других эвристик с точки зрения стоимости дерева сделаем некоторые предположения относительно пространства, из которого выбираются веса $\alpha_0, \beta_1, \alpha_1, \dots, \beta_n, \alpha_n$.

а) В случае деревьев с взвешенными внутренними вершинами (веса $\alpha_0, \alpha_1, \dots, \alpha_n$ не учитываются) веса β_1, \dots, β_n выбираются независимо из неотрицательных вещественных чисел с данным законом распределения. Если E_1 есть математическое ожидание относительно этого распределения, то тогда $E_1(\beta)$ есть ожидаемое значение отдельного веса.

б) В общем случае веса вершин β_1, \dots, β_n и веса листьев (висячих вершин) $\alpha_0, \alpha_1, \dots, \alpha_n$ независимо выбираются из двух множеств неотрицательных вещественных чисел со своими законами распределения. Пусть E_1, E_2 — математические ожидания относительно этих распределений.

Получены следующие оценки стоимости для различных классов деревьев:

— для оптимальных деревьев на множестве весов β_1, \dots, β_n

$$E_1(|T_{\text{opt}}|) = E_1(\beta)n \log n + O(n);$$

— для балансированных по числу вершин деревьев на множестве весов β_1, \dots, β_n

$$E_1(|T_{\text{bal}}|) = E_1(\beta)n \log n + O(n);$$

— для случайных деревьев на множестве весов β_1, \dots, β_n

$$E_1(|T_{\text{сл}}|) = 2 \ln 2 E_1(\beta)n \log n + O(n);$$

— для монотонных деревьев, т. е. деревьев, построенных на основании правила 1,

$$E_1(|T_m|) = 2 \ln 2 E_1(\beta) n \log n + O(n);$$

— для оптимальных деревьев в общем случае

$$\begin{aligned} E_1(|T_{opt}|) &= \\ &= E_1(\beta) n \log n + E_2(\alpha)(n+1) \log(n+1) + O(n); \end{aligned}$$

— для балансированных по числу вершин деревьев в общем случае

$$E_1(|T_-|) = E_1(\beta) n \log n + E_2(\alpha)(n+1) \log(n+1) + O(n);$$

— для случайных деревьев в общем случае

$$= 2 \ln 2 [E_1(\beta) n \log n + E_2(\alpha)(n+1) \log(n+1) + O(n)].$$

Отсюда видно, что деревья, для которых выполняется баланс по числу вершин в левом и правом поддеревьях, асимптотически так же хороши, как и оптимальные. Тот факт, что эвристика, позволяющая строить монотонные деревья в соответствии с правилом 1, дает плохие результаты, должен рассматриваться как неожиданный.

Для деревьев, балансированных по сумме весов вершин, существуют более точные результаты. Пусть $\alpha_0, \dots, \alpha_n, \beta_1, \dots, \beta_n$ — веса вершин, удовлетворяющие нормирующему условию $\sum \alpha_i + \sum \beta_j = 1$ и условию $H = \sum \beta_i \log(1/\beta_i) + \sum \alpha_i \log(1/\alpha_i)$. Тогда справедливы оценки:

а) $(1/\log 3)H \leq |T_{opt}| \leq |T_-| \leq 2 + (1 - \log(\sqrt{5} - 1))^{-1}H$
или $0,63 \leq |T_{opt}| \leq |T_-| \leq 2 + 1,44H$;

б) $H - \log H - \log e + 1 \leq |T_{opt}| \leq |T_-| \leq H + 2$.

Это означает, что бинарные деревья, балансированные по суммам весов, по стоимости всегда близки к оптимальным.

Приведем пример того, что правило 1 строит «плохие» деревья. Пусть $n = 2^k - 1$, $\beta_i = 2^{-k} + \varepsilon_i$, $\sum_{i=1}^n \varepsilon_i = 2^{-k}$, $\varepsilon_1 > \varepsilon_2 > \dots > \varepsilon_n > 0$ для $i \in [1:n]$ и $\alpha_j = 0$ для $j \in [0:n]$. Для указанных весов правило 1 строит дерево, показанное на рис. 4.9. Его стоимость удовлет-

воряет следующему неравенству:

$$|T_1| = \sum_{i=1}^n \beta_i (L_i + 1) + \sum_{j=0}^n \alpha_j L_j = \sum_{i=1}^n \beta_i i \geqslant \\ \geqslant 2^{-k} \sum_{i=1}^n i = 2^{-k} \frac{n(n+1)}{2} \geqslant \frac{n}{2}.$$

Рис. 4.10 показывает балансированное по весу дерево для тех же самых весов. Стоимость его ограничена сверху, как показано ниже, величиной $2 \log n$:

$$|T_-| = \sum_{i=1}^n \beta_i (L_i + 1) + \sum_{j=0}^n \alpha_j L_j \leqslant 2^{-(k-1)} \sum_{i=1}^n (L_i + 1) \leqslant \\ \leqslant 2^{-(k-1)} \sum_{l=1}^k 2^{(l-1)} l \leqslant 2 \log n.$$

Таким образом, $|T_1|/|T_-| > n/(4 \log n)$. Этот пример является серьезным аргументом против использования правила 1.

2.5. Конструирование динамических почти оптимальных деревьев. Рассмотренные методы построения оптимальных и почти оптимальных деревьев были основаны на предположении, что множество весов было известно заранее, поэтому было заранее известно число

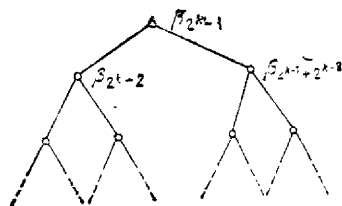


Рис. 4.10.

ло внутренних вершин. Рассмотрим теперь ситуацию, когда множество весов заранее не известно и на вход алгоритма построения дерева подается поэлементно последовательность весов $\{\beta\} = (\beta_{i_1}, \beta_{i_2}, \dots, \beta_{i_m})$ из некоторого достаточно богатого мно-

жества. Построение дерева проводится с учетом условия: если номер i_k очередного поступающего на вход алгоритма веса β_{i_k} меньше номера веса в корне, то соответствующая весу β_{i_k} вершина будет помещена в левое поддереву корня, в противном случае — в правое. Аналогично поступаем с каждым поддеревом.

Для простоты ограничимся случаем весов β , так что стоимость дерева после обработки первых N весов из последовательности $\{\beta\}$ будет иметь вид $|T| =$

$= \sum_{i=1}^N \beta_i L_i$. Изложим два подхода к решению задачи.

Первый подход основан на применении преобразования $\phi(v, T)$, описанного выше и называемого далее *простым вращением*, и преобразования, называемого *двойным вращением* и представляющего собой последовательно примененные простые вращения. Схемы обоих преобразований приведены на рис. 4.11. Эти преобразования будут применяться для локальной минимизации стоимости дерева. Второй подход основан на построении специального класса деревьев.

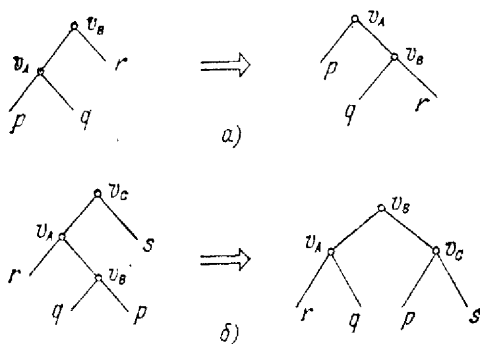


Рис. 4.11.

Пусть p, q, r и s — полные веса потомков вершин v_A, v_B и v_C (см. рис. 4.11), где под полным весом вершины понимается либо вес вершины, если она висячая, либо сумма веса вершины и полных весов ее потомков. Тогда:

— простое вращение применяется, если

$$\beta(v_A) + p > \beta(v_B) + r$$

или ($w(x)$ — полный вес вершины x)

$$w(v_A) - q > w(v_B) - w(v_A)$$

(таким образом, локально минимизируется стоимость поддерева с корнем в вершине v_A);

— двойное вращение применяется после проверки неприменимости простого вращения, если

$$2\beta(v_B) + p + q > \beta(v_C) + s$$

или

$$2w(v_B) - p - q > w(v_C) - w(v_A).$$

Пример применения указанных преобразований см. на рис. 4.12.

Статистическое исследование этой эвристики показывает, что деревья, построенные при применении только простого вращения, дают 5-процентное отклонение от оптимума, а построенные с использованием двойного вращения — 3-процентное. Можно показать,

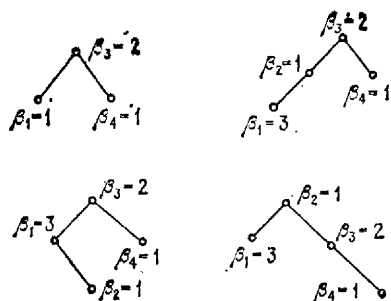


Рис. 4.12.

что при использовании только простого вращения средняя стоимость дерева над множеством весов $\beta_i = 1/n$ пропорциональна $\sqrt{\ln n}$.

2.6. Класс $B_{1/3}$ динамических деревьев. Определим следующий класс взвешенных деревьев:

а) стоимость деревьев из $B_{1/3}$ близка к оптимальной;

б) класс $B_{1/3}$ содержит в качестве подмножества множества оптимальных, балансированных по весу и так называемых минимаксных деревьев;

в) существует алгоритм A , который представляет собой последовательность локальных преобразований и который переводит дерево T_a , не принадлежащее после добавления или удаления вершины к классу $B_{1/3}$, в эквивалентное дерево T_b из класса $B_{1/3}$.

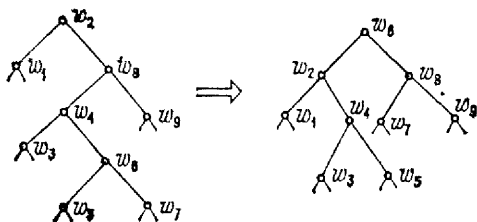


Рис. 4.13.

Локальные преобразования представляют собой уже изученные вращения (простое и двойное) и, кроме того, еще одно вращение — вращение 3 (рис. 4.13).

Пусть T — бинарное дерево, вершины которого имеют неотрицательный вес $\beta(v)$, и пусть $S(T)$ — суммар-

ный вес вершины дерева T . Тогда будем говорить, что дерево T принадлежит классу $B_{1/3}$, если:

а) T — пустое дерево;

б) $T = (T_l, v_0, T_r)$, $T_l \in B_{1/3}$, $T_r \in B_{1/3}$ и $S(T_l) \leq (2/3)S(T)$, $S(T_r) \leq (2/3)S(T)$.

Теорема 8. Пусть $T \in B_{1/3}$, и пусть $\bar{\beta} = (\beta_1, \dots, \beta_n)$ есть множество нормализованных весов, на котором построено дерево T ; тогда

$$|T| \leq H(\bar{\beta})/H(1/3, 2/3) + 1,$$

где $H(1/3, 2/3) = 1,088$.

Доказательство существенно использует следующее утверждение, которое мы приводим без доказательства.

Лемма 6. Пусть $\bar{\beta} = (\beta_1, \dots, \beta_n)$, $\beta_i \geq 0$, $1 \leq i \leq n$, $\sum_{i=1}^n \beta_i = 1$. Пусть, далее, (r_0, r_1, \dots, r_m) — подпоследовательность последовательности $(0, 1, \dots, n)$ с $r_0 = 0$, $r_m = n$,

$$\bar{\beta}^j = (\beta_{r_{j-1}+1}, \dots, \beta_{r_j}), \quad 1 \leq j \leq m, \quad \beta^j = \sum_{i=r_{j-1}+1}^{r_j} \beta_i.$$

Тогда

$$H(\bar{\beta}) = H(\beta^1, \beta^2, \dots, \beta^m) + \sum_{i=1}^m \beta^i H(\bar{\beta}^i/\beta^i).$$

Доказательство теоремы 8 будем вести индукцией по числу вершин. Для $n=1$ теорема очевидна. Предположим, что она верна для деревьев с числом вершин меньше n , и докажем ее для n . Пусть w_k — корень дерева T . Пусть $\bar{\beta}^1 = (\beta_1/s_1, \dots, \beta_{k-1}/s_1)$ и $\bar{\beta}^2 =$

$$= (\beta_{k+1}/s_2, \dots, \beta_n/s_2), \quad \text{где } s_1 = \sum_{i=1}^{k-1} \beta_i, \quad s_2 = \sum_{i=k+1}^n \beta_i$$

— нормализованные веса поддеревьев T_l , T_r . По определению стоимости дерева имеем

$$\begin{aligned} |T| &= \sum_{i=1}^n \beta_i L_i = \sum_{i=1}^{k-1} \beta_i L_i + \beta_k + \sum_{i=k+1}^n \beta_i L_i = \\ &= s_1 \left(\sum_{i=1}^{k-1} \frac{\beta_i}{s_1} (L_i - 1) \right) + s_2 \left(\sum_{i=k+1}^n \frac{\beta_i}{s_2} (L_i - 1) \right) + \sum_{i=1}^n \beta_i, \end{aligned}$$

поэтому

$$|T| = s_1 |T_l| + s_2 |T_r| + 1. \quad (4.10)$$

Пусть $d = 1/H(1/3, 2/3)$. По предположению индукции имеем

$$|T_l| \leq dH(\bar{\beta}^1) + 1, \quad |T_r| \leq dH(\bar{\beta}^2) + 1. \quad (4.11)$$

Подстановка (4.11) в (4.10) дает неравенство

$$|T| \leq d(s_1 H(\bar{\beta}^1) + s_2 H(\bar{\beta}^2)) + s_1 + s_2 + 1. \quad (4.12)$$

Из леммы 6 вытекает равенство

$$H(\bar{\beta}) = H(s_1, s_2, \beta_k) + s_1 H(\bar{\beta}^1) + s_2 H(\bar{\beta}^2),$$

которое вместе с (4.12) дает оценку сверху для стоимости:

$$|T| \leq dH(\bar{\beta}) - dH(s_1, s_2, \beta_k) + s_1 + s_2 + 1. \quad (4.13)$$

Теперь рассмотрим два случая:

а) $\beta_k > 2/3$. Тогда $s_1, s_2 < 1/3$, $H(s_1, s_2, \beta_k) \geq s_1 + s_2$, откуда, учитывая (4.13) и неравенство $d > 1$, получаем

$$|T| \leq dH(\bar{\beta}) + 1.$$

б) $\beta_k \leq 2/3$. По определению класса $B_{1/3}$ имеем $s_1 \leq 2/3$ и $s_2 \leq 2/3$. Очевидно, что

$$s_1 + s_2 \leq 2/3, \quad s_1 + \beta_k \leq 2/3, \quad s_2 + \beta_k \leq 2/3$$

(в противном случае из условий $s_1 + s_2 > 2/3$, $s_1 + \beta_k > 2/3$ и $s_2 + \beta_k > 2/3$ следовало бы неравенство $2(s_1 + s_2 + \beta_k) > 8/3$, или $2 > 8/3$, что невозможно).

Так как $s_1 + s_2 \leq 2/3$, то по лемме 6 имеем

$$H(s_1, s_2, \beta_k) \geq H(s_1 + s_2, \beta_k) \geq H(2/3, 1/3) \quad (4.14)$$

(поскольку $\beta_k \leq 2/3$, $s_1 + s_2 \leq 2/3$, $H(x, 1-x) = H(1-x, x)$ и $H(x, 1-x)$ есть монотонно возрастающая функция на отрезке $[1/2, 1]$).

Подстановка (4.14) в (4.13) дает искомое неравенство

$$|T| \leq dH(\bar{\beta}) + 1.$$

Следствие. Если $T \in B_{1/3}$ и T_{opt} — оптимальное на том же множестве весов дерево, то $|T| \leq 1,088(|T_{\text{opt}}| + \log |T_{\text{opt}}|) + 2,202$.

Для доказательства достаточно воспользоваться оценками для $|T_{\text{opt}}|$ в п. 2.4.

Теорема 9. Пусть B_{opt} — множество оптимальных, B_{min} — множество минимаксных, а B_{wb} — множе-

ство балансированных по весу деревьев; тогда

$$B_{opt} \cup B_{mm} \cup B_{wb} \subset B_{1/3}.$$

Замечание. Пусть $L(r) = \sum_{i=1}^{r-1} \beta_i$, $R(r) = \sum_{i=r+1}^n \beta_i$, $1 \leq r \leq n$, и пусть w — номер вершины, являющейся корнем. Тогда $T = (T_l, v_w, T_r)$ называется *балансируемым по весу* деревом, $T \in B_{wb}$, если $T_l \in B_{wb}$, $T_r \in B_{wb}$ и $|L(w) - R(w)| \leq |L(r) - R(r)|$ для всех $r \in [1 : n]$. Дерево T называется *минимаксным* деревом, $T \in B_{mm}$, если $T_l \in B_{mm}$, $T_r \in B_{mm}$ и $\max(L(w), R(w)) \leq \max(L(r), R(r))$ для всех $r \in [1 : n]$.

Доказательство теоремы 9. Пусть $T \in B_{mm}$; тогда из неравенств

$$S(T_l) = \sum_{x \in T_l} \beta(x) \leq \frac{1}{2} S(T),$$

$$S(T_r) \leq (1/2)S(T)$$

следует, что $T \in B_{1/3}$.

Пусть теперь $T \in B_{opt}$ имеет вид, показанный на рис. 4.14, и пусть s_i есть сумма весов вершин дерева с корнем в вершине w_i . Предположим, что $T \notin B_{1/3}$; тогда $S(T_r) > (2/3)S(T)$. Рассмотрим два случая:

а) $\beta(w_8) + s_9 > (1/3)S(T)$. В этом случае дерево T' (рис. 4.15) эквивалентно по множеству весов дереву T и $|T'| < |T|$, а это противоречит оптимальности дерева T .

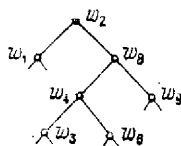


Рис. 4.14.

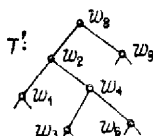


Рис. 4.15.

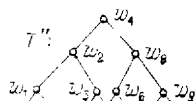


Рис. 4.16.

б) $\beta(w_8) + s_9 \leq (1/3)S(T) \Rightarrow s_4 > (1/3)S(T)$. Теперь к противоречию приводит дерево T'' на рис. 4.16.

Рассмотрим дерево T , показанное на рис. 4.17 и принадлежащее классу B_{wb} . Пусть T_l — поддерево с корнем w_l . Положим $s_l = S(T_l)$, $\beta_l = \beta(w_l)$ и предположим, что $T \notin B_{1/3}$. Из этого предположения вытекает

неравенство $s_1 > (2/3)s_2$. Пусть w_x — вершина из T_1 с весом $\beta(w_x) > 0$ и такая, что для всех $w \in T_1$ из $\beta(w) > 0$ следует $w < w_x$, т. е. все вершины w принадлежат левому поддереву

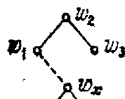


Рис. 4.17.

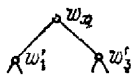


Рис. 4.18.

дерева с корнем в вершине w_x . Рассмотрим теперь дерево с корнем в вершине w_x , но эквивалентное по множеству весов дере-

ву T (рис. 4.18). Пусть T'_i — дерево с корнем w'_i , $s'_i = S(T'_i)$; тогда

$$s'_1 = s_1 - \beta_x, \quad (4.15)$$

$$s'_3 = s_3 + \beta_x, \quad (4.16)$$

$$T \in B_{w_b} \Rightarrow |s'_1 - s'_3| \geq s_1 - s_3. \quad (4.17)$$

Рассмотрим два случая:

а) $s'_1 \geq s'_3$. В этом случае (4.15)–(4.17) приводят к неравенству $s_1 - \beta_x - s_3 - \beta_x \geq s_1 - s_3$, или $\beta_x + \beta_x \leq 0$, что противоречит неравенству $\beta_x > 0$.

б) $s'_1 < s'_3$. В этом случае (4.15)–(4.17) приводят к неравенствам $s_3 + \beta_x - s_1 + \beta_x \geq s_1 - s_3$, $2(s_3 + \beta_x) + \beta_x \geq 2s_1 + \beta_x$, а так как $2(s_3 + \beta_x) < (1/3)s_2$ и $2s_1 > (2/3)s_2$ в силу неравенства $s_1 > (2/3)s_2$, то отсюда в свою очередь следует

$$\beta_x > (2/3)s_2 + \beta_x \Rightarrow s_1 - \beta_x < (1/3)s_2.$$

Из неравенства $s_1 > (2/3)s_2$ следует $\beta_x + s_3 < (1/3)s_2$ и

$$s_1 - s_3 > (1/3)s_2. \quad (4.18)$$

С другой стороны, из неравенств $s_1 - \beta_x < (1/3)s_1$ и $\beta_x + s_3 < (1/3)s_2$ вытекает

$$|s_3 + \beta_x - s_1 + \beta_x| = |s'_1 - s'_3| < (1/3)s_2. \quad (4.19)$$

Нетрудно заметить, что неравенства (4.18) и (4.19) противоречат неравенству (4.17). Теорема доказана.

Опишем на содержательном уровне алгоритм, который гарантирует, что после включения или удаления вершины результирующее дерево будет снова принадлежать классу $B_{1/3}$. Алгоритм состоит из двух частей. Первая часть включает вершину в дерево или удаляет вершину из дерева и хранит путь от корня к включае-

мой или удаляемой вершине в стеке S . Кроме того, она приводит в порядок суммы s_i весов вершин на пути из корня к включаемой или удаляемой вершине. Вторая часть проверяет, является ли дерево, полученное в результате работы первой части, элементом класса $B_{1/3}$ или нет, и над деревом производятся соответствующие операции, пока дерево не станет удовлетворять требованиям класса $B_{1/3}$. Во время работы этой части справедлив следующий инвариант: если вершина находится в вершине стека S , поддеревья этой вершины суть элементы класса $B_{1/3}$.

Первая часть алгоритма работает следующим образом.

Если мы включаем вершину, то мы храним последовательность вершин от корня до включаемой вершины в стеке S .

Если мы удаляем вершину, то храним последовательность вершин от корня к удаляемой вершине, но без нее, в стеке S .

Теперь рассмотрим два случая:

а) Вершина w , удаляемая из дерева, есть висячая вершина или она имеет только одно поддерево. Тогда мы удаляем вершину w и подсоединяем ее поддерево на освободившееся место к предку удаленной вершины w ; стек S остается без изменения.

б) Вершина w имеет два поддерева. Тогда, начиная с вершины, являющейся корнем правого поддерева вершины w , движемся вниз последовательно по левым поддеревьям, пока не достигнем вершины w' , не имеющей левого поддерева. Занесем вершины этого пути в вспомогательный список AL (без w'). Поменяем местами вершины w и w' и удалим w , как это делается в случае а). Потом добавим вершину w' в стек S . Теперь включим в стек вершины из списка AL в том же самом порядке, в котором они были занесены в список AL .

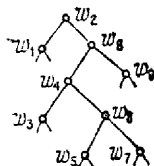


Рис. 4.19.

Вторая часть состоит из следующих шагов.

Шаг 1. Если стек S пуст, то алгоритм заканчивается. В противном случае пусть w_2 — вершина стека S и корень дерева, показанного на рис. 4.19.

Введем следующие обозначения: $\beta_i = \beta(w_i)$, T_i — дерево с корнем w_i (до вращения), $s_i = S(T_i)$.

Возможны четыре случая:

а) $s_1 + \beta_2 \geq (1/3)s_2$ и $\beta_2 + s_3 \geq (1/3)s_2$ — переход к шагу 2;

б) $s_1 + \beta_2 < (1/3)s_2$ и $\beta_3 + s_3 \geq (1/3)s_2$ — переход к шагу 3;

в) $s_1 + \beta_2 < (1/3)s_2$, $\beta_3 + s_3 < (1/3)s_2$ и $s_1 + \beta_2 + s_3 + \beta_4 \geq (1/3)s_2$ — переход к шагу 4;

г) $\beta_3 + s_3 \leq (1/3)s_2$ и $s_1 + \beta_2 + s_3 + \beta_4 < (1/3)s_2$ — переход к шагу 5.

Шаг 2. Удалить вершину w_2 из стека S и перейти к шагу 1.

Шаг 3. Провести простое вращение и перейти к шагу 1.

Шаг 4. Провести двойное вращение, включить вершину w_3 в стек и перейти к шагу 1.

Шаг 5. Провести вращение 3 и ввести вершину w_3 в стек S . Если $\beta_4 + s_3 \geq (1/3)(s_2 + \beta_4 + s_3)$, то перейти к шагу 1; если $s_1 + \beta_2 \geq (1/3)(s_1 + \beta_2 + s_3 + \beta_4 + s_3)$, то удалить вершину w_2 из стека S , ввести вершину w_4 в стек S , перейти к шагу 1; в противном случае применить ряд дополнительных преобразований к поддереву с корнем w_2 , удалить вершину w_2 из стека S и перейти к шагу 1.

Примечание. Мы не уточняем суть дополнительных преобразований в шаге 5, так как этот шаг, по свидетельству автора алгоритма, нужен для оценки трудоемкости алгоритма и на практике встречается чрезвычайно редко; в экспериментах на массиве из нескольких тысяч деревьев он не встретился ни разу.

Теорема 10. Пусть $T \in B_{1/3}$, и пусть T_a — дерево, полученное из T в результате включения или удаления вершины. Тогда применение описанного алгоритма преобразует дерево T_a в дерево T_b из класса $B_{1/3}$.

Доказательство. Используем инвариант, упомянутый в описании алгоритма: если вершина находится в вершине стека, то поддеревья дерева с корнем в этой вершине принадлежат классу $B_{1/3}$.

Пусть T_2 — дерево с корнем w_2 до вращения. Тогда $T_{12} \in B_{1/3}$ и $T_{r2} \in B_{1/3}$. Пусть T' — то же дерево после вращения, и пусть w' — корень дерева T' ; докажем, что

$$S(T'_l) + \beta(w') \geq (1/3) S(T') = (1/3) s_2,$$

$$S(T'_r) + \beta(w') \geq (1/3) S(T') = (1/3) s_{21}$$

т. е. T' удовлетворяет условиям определения класса $B_{1/3}$. Как только мы поместим корни поддеревьев, которые не проверяются, в стек, теорема будет доказана.

Рассмотрим четыре перечисленных при описании алгоритма случая.

а) Неравенства $s_1 + \beta_2 \geq (1/3)s_2$ и $s_3 + \beta_2 \geq (1/3)s_2$ верны в силу исходного предположения.

б) Покажем, что

$$\beta_3 + s_9 \geq (1/3)s_2, \quad (4.20)$$

$$s_1 + \beta_2 + s_4 + \beta_8 \geq (1/3)s_2. \quad (4.21)$$

Но (4.20) верно по предположению, а (4.21) следует из неравенств $s_9 \leq (2/3)s_8 \leq (2/3)s_2$.

в) Покажем, что

$$s_1 + \beta_2 + s_3 + \beta_4 \geq (1/3)s_2, \quad (4.22)$$

$$\beta_4 + s_6 + \beta_8 + s_9 \geq (1/3)s_2. \quad (4.23)$$

Но (4.22) верно по предположению, а (4.23) следует из неравенств $s_3 \leq (4/9)s_8$ и $s_8 > (2/3)s_2$:

$$\beta_4 + s_6 + \beta_8 + s_9 =$$

$$= s_8 - s_3 \geq (5/9)s_8 > (5/9)(2/3)s_2 > (1/3)s_2.$$

Первое из неравенств вытекает в свою очередь из неравенств $s_3 \leq (2/3)s_4$ и $s_4 \leq (2/3)s_8$, которые являются следствиями принадлежности деревьев T_4 и T_8 классу $B_{1/3}$. Второе неравенство вытекает из неравенства $s_1 + \beta_2 < (1/3)s_2$.

г) Покажем, что

$$s_2 + \beta_2 + s_3 + \beta_4 + s_5 + \beta_6 \geq (1/3)s_2, \quad (4.24)$$

$$\beta_6 + s_7 + \beta_8 + s_9 \geq (1/3)s_2, \quad (4.25)$$

$$s_4 + \beta_4 \geq (1/3)(s_3 + \beta_4 + s_5). \quad (4.26)$$

$$s_8 + \beta_4 + s_5 \geq (1/3)(s_1 + \beta_2 + s_3 + \beta_4 + s_5). \quad (4.27)$$

Неравенство (4.24) вытекает из равенства

$$s_1 + \beta_2 + s_3 + \beta_4 + s_5 + \beta_6 = s_2 - (\beta_8 + s_9) - s_7, \quad (4.28)$$

если учесть, что $\beta_8 + s_9 < (1/3)s_2$, $s_7 \leq (2/3)s_8$ (так как $T_8 \in B_{1/3}$), $s_6 \leq (2/3)s_4$ (так как $T_4 \in B_{1/3}$), $s_4 \leq (2/3)s_8$ (так как $T_8 \in B_{1/3}$), $s_7 \leq (8/27)s_8 < (8/27)s_2$ (в силу трех предыдущих неравенств). Подстановка этих неравенств

в (4.28) дает

$$s_1 + \beta_2 + s_3 + \beta_4 + s_5 + \beta_6 > \\ > s_2 - (1/3)s_2 - (8/27)s_2 > (1/3)s_2.$$

Неравенство (4.25) вытекает из равенства

$$\beta_6 + s_7 + \beta_8 + s_9 = s_2 - (s_1 + \beta_2 + s_3 + \beta_4) - s_5, \quad (4.29)$$

если учесть, что $s_1 + \beta_2 + s_3 + \beta_4 < (1/3)s_2$, $s_5 \leq (8/27)s_2 \leq (8/27)s_2$ (так как $T_6, T_4, T_3 \in B_{1/3}$). Подстановка этих неравенств в (4.29) дает искомое:

$$\beta_6 + s_7 + \beta_8 + s_9 > s_2 - (1/3)s_2 - (8/27)s_2 \geq (1/3)s_2.$$

Неравенство (4.26) непосредственно следует из того, что $T_4 \in B_{1/3}$.

Для доказательства неравенства (4.27) заметим, что неравенства $\beta_8 + s_9 < (1/3)s_2$, $s_1 + \beta_2 + s_3 + \beta_4 < (1/3)s_2$ дают вместе неравенство

$$s_4 > (1/3)s_2, \quad (4.30)$$

которое вместе с условием $T_4 \in B_{1/3}$ приводит к неравенству

$$s_3 + \beta_4 > (1/6)s_2. \quad (4.31)$$

Из (4.31) и неравенства $s_1 + \beta_2 + s_3 + \beta_4 < (1/3)s_2$ вытекает неравенство $s_1 + \beta_2 < (1/6)s_2$; из него и неравенства (4.31) следует $s_3 + \beta_4 < s_1 + \beta_2$, откуда и вытекает (4.27). Теорема доказана.

Без доказательства укажем, что при достаточно слабых предположениях о законе распределения весов среднее число операций в случае включения вершины меньше чем $O(l)$, где l — число вершин в пути от корня до включенной вершины.

2.7. Построение оптимальных каркасов. Пусть имеется неориентированный граф $G = (V, E)$, каждому ребру которого сопоставлен вес (или длина) $l(e)$, $e \in E$. Простейшей задачей о построении оптимального каркаса является задача о кратчайшей связывающей сети, или задача нахождения каркаса с наименьшей суммой весов входящих в него ребер. Эта задача хорошо известна; напомним здесь два алгоритма ее решения: алгоритм Краскала и алгоритм Прима.

Первый состоит в упорядочении ребер графа по убыванию (или возрастанию) их длин. После этого ребра графа рассматривают последовательно и либо

удаляют ребро из графа, если не нарушается связность, либо оставляют, если связность нарушается. Нетрудно увидеть, что в результате останется ровно $|V| - 1$ ребер; совокупность их определяет связный суграф с тем же множеством вершин, что и исходный граф, т. е. каркас. Минимальность суммы весов ребер вытекает из построения.

Второй алгоритм состоит в последовательном наращивании фрагмента или фрагментов (вначале это — отдельные вершины) за счет добавления ребер, дающих наименьшее приращение стоимости фрагмента (суммы весов ребер) и не приводящих к образованию циклов. Алгоритм является реализацией метода динамического программирования и обладает высокой эффективностью.

В общем случае, когда зависимость стоимости дерева от весов ребер имеет более сложный вид, для отыскания оптимального каркаса неизбежно применение полного перебора. Для этой цели пригоден подход, основанный на процедурах генерирования всех каркасов графа, в частности на процедуре быстрого перебора всех частичных графов с заданным цикломатическим числом. Изложим основную идею этой процедуры.

λ -графом данного графа $G = (V, E)$ называется его частичный граф $G^\lambda = (X, E')$; $E' \subset E$, с цикломатическим числом λ , имеющий столько же компонент связности, что и граф G .

Предлагаемый ниже алгоритм требует упорядоченного перебора $C_{|E|}^{M_\lambda}$ сочетаний ребер, где $M_\lambda = |V| + \lambda - p$, с проверкой сохранения числа p компонент связности в каждом частичном графе. Перенумеруем в произвольном порядке ребра исходного графа. Нам нужно для получения каркаса исключить $|E| - M_\lambda$ ребер. Если при исключении первого ребра число компонент связности не изменяется, то переходим ко второму ребру, и т. д. Пусть на некотором шаге исключено какое-то ребро. Попробуем исключить еще одно ребро, начиная с номера, на единицу большего номера последнего ребра, помещенного в стек. Действуя так $|E| - M_\lambda$ раз, получим первый λ -граф. В стек при этом окажутся ребра, исключенные из исходного графа. Для того чтобы найти очередной λ -граф, исключим из стека последнее загруженное ребро и вернем это ребро в граф. Попробуем исключить из графа

ребро с номером, на единицу большим номера ребра, только что присоединенного к графу. Если это возможно, то получим очередной λ -граф, если нет — перейдем к следующему ребру, и т. д. Если мы достигнем ребра с наибольшим номером и ни одно из ребер нельзя исключить, то достанем из стека последнее загруженное в него ребро и вернем это ребро в граф. Попробуем исключить из графа ребро с номером, на единицу большим номера ребра, только что присоединенного к графу, и т. д. Каждый раз, когда сформирован очередной искомый граф, в стеке оказывается $|E| - M_\lambda$ исключенных ребер.

Действуя таким образом, можно перебрать все λ -графы. Процесс закончится при попытке достать ребро из пустого стека. Алгоритм организован так, что если частичная конфигурация исключенных ребер изменяет число компонент связности, то автоматически игнорируются все «висящие» на ней конфигурации.

§ 3. Балансированные деревья

В этом параграфе мы рассмотрим деревья, критерием оптимальности которых служат некоторые структурные характеристики. В случае АВЛ-деревьев такой характеристикой служит разность между высотами правого и левого поддеревьев для каждой вершины, в случае ВВ-деревьев — некоторая функция, отражающая соотношение числа вершин в правом и левом поддеревьях для каждой вершины.

3.1. Балансированные по высоте деревья, или АВЛ-деревья. Бинарное дерево называется *балансированным по высоте* или *АВЛ-деревом*, если для любой его вершины высота правого поддерева отличается от высоты левого поддерева не более чем на единицу. (Высота поддерева есть разность между наибольшим из уровней его вершин и уровнем его корня).

Теорема 11. *Наибольшая длина ветвей в n -вершинном АВЛ-дереве не превышает $(3/2) \log n$.*

Доказательство. Наиболее асимметричное АВЛ-дерево T_n высоты h имеет наиболее асимметричное АВЛ-дерево T_{h-1} высоты $h-1$ в качестве одного из своих поддеревьев и наиболее асимметричное АВЛ-дерево T_{h-2} высоты $h-2$ в качестве другого (как показано на рис. 4.20).

Обозначая число вершин в T_h через n_h , получаем рекуррентное соотношение

$$n_h = n_{h-1} + n_{h-2} + 1, \quad n_{-1} = 0, \quad n_0 = 1. \quad (4.32)$$

Для $h=3$ и $h=4$ можно непосредственно проверить, а затем по индукции доказать, что при $h > 3$ справедливо неравенство $n_h > \alpha^{h+1}$, где $\alpha = (\sqrt{5} + 1)/2$ есть положительный корень уравнения $x^2 = x + 1$. Следовательно, число вершин n удовлетворяет неравенству $n \geq n_h > \alpha^{h+1}$, откуда $h + 1 < \log 2 \cdot \log n \approx 1,44 \log n$.

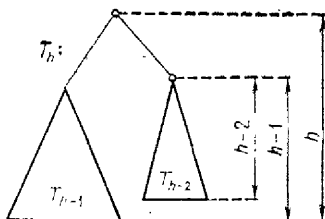


Рис. 4.20.

Но $h + 1$ — это число вершин в пути длины h из корня в некоторую вершину. Следовательно, для AVL-дерева это число никогда не превышает $1,44 \log n$. Для случайного дерева средняя длина пути из корня в всякую вершину составляет $\sim 1,39 \log n$, а в худшем случае она может оказаться равной n . Таким образом, худший случай для AVL-дерева ненамного уступает средним возможностям случайного дерева. В среднем для AVL-дерева можно получить оценку $c \log n$ (для больших n), где

$$c = \frac{\alpha}{\sqrt{5} \log \alpha} \approx 1,04.$$

Для получения этой оценки обозначим через P_h суммарную длину внутренних путей в дереве T_h высоты h . Для P_h имеет место следующее рекуррентное соотношение, вытекающее из рассмотрения наиболее асимметричного AVL-дерева высоты h (рис. 4.20):

$$P_h = P_{h-1} + P_{h-2} + n_h - 1, \quad P_{-1} = P_0 = 0. \quad (4.33)$$

Деление этого соотношения на n_h дает

$$\frac{P_h}{n_h} = \frac{n_{h-1}}{n_h} \cdot \frac{P_{h-1}}{n_{h-1}} + \frac{n_{h-2}}{n_h} \cdot \frac{P_{h-2}}{n_{h-2}} + 1 - \frac{1}{n_h},$$

и, следовательно,

$$S_h = \frac{n_{h-1}}{n_h} S_{h-1} + \frac{n_{h-2}}{n_h} S_{h-2} + 1 - \frac{1}{n_h}.$$

Опуская член $(-1/n_h)$ и используя приближенное зна-

чение $n_h \approx 1,9 \left(\frac{1 + \sqrt{5}}{2} \right)^h$ для замены n_{h-1}/n_h и n_{h-2}/n_h их приближениями $(2/\sqrt{5} + 1)$, $1/\sqrt{(1 + \sqrt{5})/2}$ соответственно, получаем

$$S_h = \left(\frac{1 + \sqrt{5}}{2} \right)^{-1} S_{h-1} + \left(\frac{1 + \sqrt{5}}{2} \right)^{-2} S_{h-2} + 1.$$

Решая это уравнение, получаем

$$S_h \approx \frac{5 + \sqrt{5}}{10} h + \hat{c} + O(1) \approx \frac{5 + \sqrt{5}}{10} h, \quad h \rightarrow \infty;$$

подставляя значение для h , окончательно имеем

$$S_h \approx \frac{5 + \sqrt{5}}{10 \log [(1 + \sqrt{5})/2]} \log n_h \approx 1,04 n_h.$$

Проведенный анализ показывает, что балансированные по высоте деревья имеют достаточно короткие пути из корня в всякие вершины, что позволяет использовать их для организации поиска в больших массивах информации. Чтобы окончательно убедиться в этом, нужно показать, что включение и исключение вершины можно проводить, оставляя деревья балансированными.

3.2. Включение новой вершины в AVL-дерево. Будем характеризовать каждую вершину дерева парами 00, 01 или 10, смотря по тому, равны ли высоты правого и левого поддеревьев для данной вершины, больше высота правого поддерева или наоборот. При этом будем считать, что если вершина не имеет с какой-нибудь стороны потомка, то длина ветви с этой стороны равна нулю.

Включение новой вершины в AVL-дерево происходит следующим образом. Вначале мы подвешиваем новую вершину так, чтобы дерево оставалось бинарным, и приписываем ей характеристику 00. Следующий шаг состоит в исправлении изменившихся характеристик вершин дерева, с изменением, если нужно, структуры дерева. Это исправление касается только вершин, лежащих на пути от корня до новой вершины. Для определенности будем называть этот путь *отмеченным*. Прежде всего скорректируем характеристику предка w новой вершины. Имеются всего две возможности: а) характеристика w равна 00, и б) характеристика w равна 01 (для определенности пола-

гаем, что новая вершина была подвешена к w слева). В первом случае мы изменяем 00 на 01 или 10, смотря по тому, с какой стороны была подвешена вершина. Во втором случае новая вершина добавляется к вершине w со свободной стороны, при этом длины обеих ветвей, исходящих из w , становятся равными (равными 1), поэтому замена 01 на 00 завершает включение новой вершины.

Рассмотрим структуру отмеченного пути, когда вершина w имеет характеристику 00.

Лемма 7. *Подпуть Q отмеченного пути P , во всех вершинах которого характеристики равны 00, а в первой (ближайшей к корню) характеристика отлична от 00, имеет длину, равную высоте поддерева с корнем в первой вершине подпути Q .*

Доказательство. Допустим, что наше утверждение неверно. Обозначим первую вершину подпути Q (в которой характеристика отлична от 00) через v . Из всей совокупности путей, начинающихся в v и имеющих длину, большую чем длина пути Q , выберем путь R , имеющий с Q наибольшее число общих вершин. Пусть t — последняя общая вершина путей Q и R . Тогда ее ветви имеют разную длину; следовательно, характеристика вершины t не может быть равна 00. Лемма доказана.

Из леммы 7 вытекает, что корректировка характеристик приведет к изменению характеристик вершин подпути Q с 00 на 01 или 10 и при этом любое поддерево с корнем в вершине остается сбалансированным. Возможны три случая:

1) $Q = P$. Тогда все дерево остается сбалансированным.

2) Подпуть Q лежит в короткой ветви поддерева $T(v)$. Тогда после добавления новой вершины получилось сбалансированное дерево, и характеристику вершины v нужно только заменить на 00.

3) Подпуть Q лежит в длинной ветви поддерева $T(v)$. Если при этом последние два шага при движении к v осуществлялись в одном направлении, т. е. оба перехода были от правого потомка к предку или оба — от левого, то выполняется простое вращение (рис. 4.21). Если два последних шага выполнялись в разных направлениях, то выполняется двойное вращение (рис. 4.22). На этих рисунках нарушение баланса происходит в вершине v , т. е. первой вершине

подпути Q . Можно проверить, что условия на высоты поддеревьев до и после преобразований выполнены и

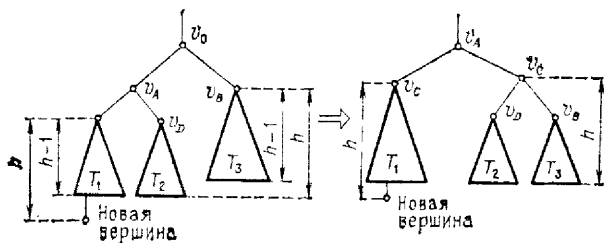


Рис. 4.21.

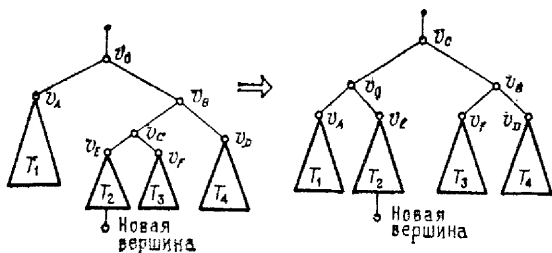


Рис. 4.22.

что после этих локальных преобразований в целом дерево является сбалансированным по высоте.

3.3. Удаление вершины из AVL-дерева. Удаление вершины принципиально не отличается от процедуры включения, уточнения требуют только детали. Преобразованием типа $\phi(v, T)$ удаление любой вершины можно свести к удалению вершины с одним пустым поддеревом. Если исключаемая вершина имеет одно пустое поддерево, то, согласно ограничению на высоты

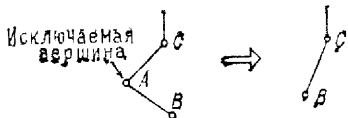


Рис. 4.23.

деревьев, другое поддерево состоит из одной вершины, как показано на рис. 4.23. В этом случае влияние на сбалансированность дерева такое же, как если бы исключалась

висячая вершина. Следовательно, имеет смысл рассмотреть только этот последний случай.

Когда исключается висячая вершина, поддерево, состоящее именно из этой вершины, теряет единицу

высоты. Снова пройдем отмеченный путь — путь от удаленной вершины до корня, проверяя в каждой вершине, какое влияние может оказать укорачивание поддерева. Преобразование, которое нужно применить зависит от характеристики вершины до удаления, от направления последнего шага и в некоторых случаях от условия на высоту потомка вершины. (В описанных ниже случаях штриховые линии обозначают укорачиваемое поддерево.) В некоторых случаях преобразование восстанавливает высоту поддерева до той, какая была до исключения; если это не так, то алгоритм заканчивается, поскольку исключение не влияет на характеристики всех расположенных выше вершин. В других случаях сведения о том, что поддерево укорачивалось, должно быть продвинуто вверх по пути к корню.

1. Если текущая вершина имеет характеристику 00, то укорачивание дерева не влияет на высоту поддерева, имеющего корнем текущую вершину. Алгоритм заканчивает свою работу (рис. 4.24).

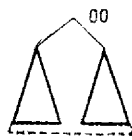
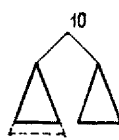
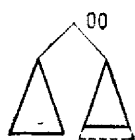


Рис. 4.24.

Рис. 4.25.

2. Если текущая вершина имеет характеристику 01 или 10 и последний шаг начинался из более высокого из двух поддеревьев текущей вершины, то продолжается движение вверх, при этом продвигается информация о том, что дерево укоротилось (рис. 4.25).

3. Если текущая вершина имеет характеристику 01 или 10 и последний шаг начинался из более низкого из двух поддеревьев текущей вершины, то в текущей вершине ограничение на высоту поддеревьев нарушается. Различаются три подслучая и их зеркальные отражения в соответствии с условием на высоту в другом потомке текущей вершины.

а) Подходящее простое вращение восстанавливает ограничение на высоту поддеревьев в текущей вершине без изменения высоты поддерева, имеющего корнем текущую вершину. Алгоритм заканчивает работу (рис. 4.26).

б) Подходящее простое вращение восстанавливает ограничение на высоту поддеревьев в текущей вершине. Продолжается движение вверх, при этом продвигается информация о том, что поддерево, имеющее корнем текущую вершину, укоротилось (рис. 4.27).

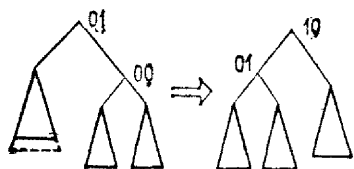


Рис. 4.26.

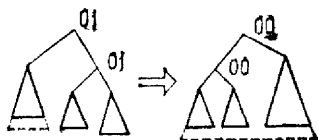


Рис. 4.27.

в) Подходящее двойное вращение восстанавливает ограничение на высоту поддеревьев в текущей вершине.

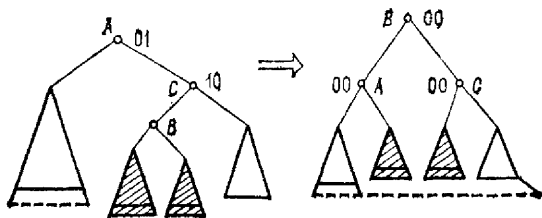


Рис. 4.28.

Продолжается движение вверх, при этом продвигается информация о том, что поддерево, имеющее корнем текущую вершину, укоротилось (рис. 4.28).

3.4. Балансированные по весу деревья (BB-деревья). Это класс бинарных деревьев, в которых ограничение на высоты поддеревьев заменено ограничением на число вершин в поддеревьях. Хотя они базируются на других принципах и не сравнимы с AVL-деревьями — поскольку эти классы не содержат друг друга и практически не пересекаются, — они обладают схожими свойствами. От AVL-деревьев BB-деревья отличаются тем, что содержат параметр, который может изменяться так, что можно произвольно ограничивать длину самого длинного пути из корня в всякую вершину за счет увеличения дисбаланса.

Пусть $T_n = (T_l, v, T_r)$ есть бинарное дерево с корнем v , где T_l и T_r — бинарные поддеревья с n_l и n_r

являющееся полностью балансированным бинарным деревом. Дерево T' должно иметь два полностью балансированных поддерева T_l и T_r с $l = 2^s - 1$ и $r = 2^t - 1$ вершинами соответственно, причем $s \neq t$. Предположим, что $s < t$. Тогда корневой баланс дерева T' равен $\beta(T') = 1/(1 + 2^{t-s}) \leq 1/3$, откуда следует, что T не может принадлежать $BB[\alpha]$ для любого $\alpha > 1/3$.

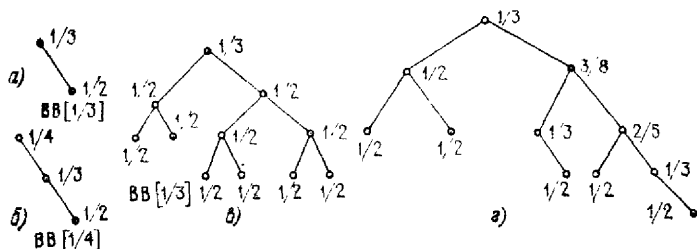


Рис. 4.29.

Теорема 13. Для любого α , $0 < \alpha < 1/2$, существуют в $BB[\alpha]$ деревья, не являющиеся AVL-деревьями.

Для доказательства достаточно указать пример такого дерева. Им может служить, например, дерево на рис. 4.29, г, которое принадлежит классу $BB[1/3]$, но не является балансированным по высоте. Аналогичным образом может быть построено такое дерево для заданного α .

Теорема 14. Для любого α , $0 < \alpha < 1/2$, существуют AVL-деревья, не принадлежащие $BB[\alpha]$.

Доказательство. Рассмотрим дерево T , левое поддерево которого является наиболее асимметричным балансированным по высоте деревом высоты h , показанным на рис. 4.20, с n_h вершинами, где $n_h \approx 1,9((1 + \sqrt{5})/2)^h$, а правое поддерево которого есть полностью балансированное бинарное дерево высоты h с $2^{h+1} - 1$ вершинами. Дерево T балансировано по высоте, но $\beta(T) \rightarrow 0$ при $h \rightarrow \infty$.

Из теорем 13 и 14 следует, что балансированность по весу и балансированность по высоте являются независимыми критериями балансированности.

Теорема 15. Высота дерева T_n из класса $BB[\alpha]$ не превышает

$$(\log(n+1) - 1) / \log(1/(1 - \alpha)).$$

• Чтобы получить эту оценку, рассмотрим наиболее асимметричное n -вершинное дерево T_n из класса

$BB[\alpha]$ («наиболее асимметричное» в данном случае означает дерево совершенно другой структуры, нежели в случае АВЛ-деревьев). Это дерево имеет $\alpha(n+1) - 1 \approx \alpha n$ вершин в одном из его поддеревьев, например в левом, и $(1-\alpha)(n+1) - 1 \approx (1-\alpha)n$ вершин в другом. Оба поддерева являются наиболее асимметричными; отсюда следует, что высота $h_\alpha(n)$ этого наиболее асимметричного дерева T_n в $BB[\alpha]$ удовлетворяет рекуррентному соотношению

$$h_\alpha(n) = 1 + h_\alpha[(1-\alpha)(n+1) - 1] \approx 1 + h_\alpha[(1-\alpha)n]$$

с решением

$$h_\alpha(n) \approx \log(n+1)/\log(1/(1-\alpha)) + O(1).$$

Для полностью балансированных бинарных деревьев ($\alpha = 1/2$) эта формула дает $h_{1/2}(n) \approx \log(n+1)$. Для $\alpha = 1 - \sqrt{2}/2$ — наибольшего значения α , для которого работает описываемый ниже алгоритм перестройки, имеем

$$h_{1-\sqrt{2}/2}(n) \approx 2 \log(n+1) + O(1).$$

Таким образом, длина самого длинного пути из корня в висячую вершину — высота — наиболее асимметричного дерева из $BB[1 - \sqrt{2}/2]$ лишь в два раза больше высоты полностью балансированного дерева.

Что касается средней высоты, то в принципе она может быть вычислена аналогично тому, как это делалось для АВЛ-деревьев, если принять во внимание, что суммарная длина внутренних путей в T_n удовлетворяет неравенству

$$|T_n| \leq \frac{1}{H(\alpha)} (n+1) \log(n+1) - 2n,$$

где $H(\alpha) = -\alpha \log \alpha - (1-\alpha) \log(1-\alpha)$.

3.5. Включение новой вершины в BB -дерево. Сопоставим каждой вершине BB -дерева число $\bar{S}(x)$, равное числу вершин в поддереве $T(x)$ с корнем в вершине x . Пусть μ есть путь из корня во вновь включенную вершину w BB -дерева. Двигаясь по пути μ в направлении от корня, проверяем на каждом шагу, нарушит ли добавление вершины баланс дерева; если нет, то добавляем единицу к $\bar{S}(x)$ и продолжаем движение. Если же добавление вершины нарушает баланс, то

прежде чем двигаться дальше, проводим необходимые преобразования, тем более что они те же самые, что и для АВЛ-деревьев.

Алгоритм включения новой вершины в ВВ-дерево:

начало

1. найти путь μ из новой вершины w в корень;

2. упорядочить вершины пути μ в направлении от корня; пусть это будет последовательность $v_{i_1}, v_{i_2}, \dots, v_{i_k}$;

3. для j от 1 до k шаг 1 цикл

4. **начало**

5. вычисляем баланс v вершины v_{i_j} по формуле

$$v = \frac{\bar{S}(\text{левый потомок } v_{i_j}) + 1}{\bar{S}(v_{i_j}) + 2};$$

6. если w включена в левое поддереву дерева $T(v_{i_j})$ то

7. если $\alpha \leq v \leq 1 - \alpha$ то

8. $\bar{S}(v_{i_j}) := \bar{S}(v_{i_j}) + 1$;

9. иначе если $\bar{S}(v_{i_j}) = 2$ то

10. использовать вращение без учета вершины w ;

11. иначе **начало** вычислить значение β — баланс правого поддерева — с учетом новой вершины w ;

12. если $1 - \beta < (1 - 2\alpha)/(1 - \alpha)$ то

13. применить простое вращение

14. иначе применить двойное вращение;

15. **конец**

16. иначе если $\alpha \leq v \leq 1 - \alpha$ то

17. $\bar{S}(v_{i_j}) := \bar{S}(v_{i_j}) + 1$;

18. иначе если $\bar{S}(v_{i_j}) = 2$ то

19. использовать простое вращение без учета вершины w ;

20. иначе вычислить значение β — баланс левого поддерева — с учетом новой вершины w ;

21. если $\beta < (1 - 2\alpha)/(1 - \alpha)$ то

22. применить простое вращение

23. иначе применить двойное вращение;

24. **конец цикла**;

конец

Схемы простого и двойного вращений с указанием новых балансов показаны на рис. 4.30.

Пример включения вершины w в дерево, показанное на рис. 4.29, g , дан на рис. 4.31.

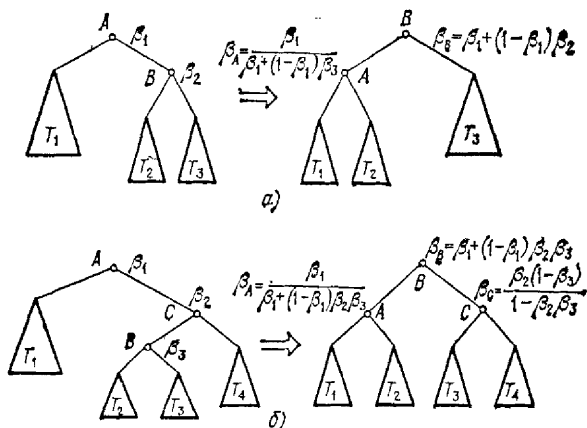


Рис. 4.30.

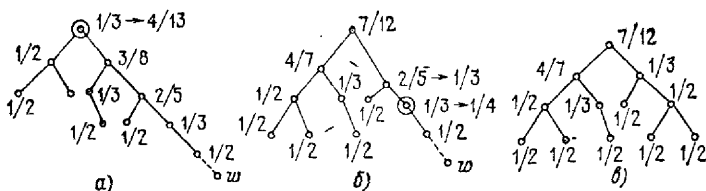


Рис. 4.31.

Удаление вершины из BB -дерева происходит аналогично включению. Следуя по пути μ из корня в вершину, подлежащую удалению, уменьшаем на единицу значение $\bar{S}(x)$ для каждой пройденной вершины. Если при этом дерево становится несбалансированным, то применяем соответствующее преобразование и движемся дальше. Когда мы достигаем вершины x_N , которую намерены удалить, может встретиться один из трех случаев. Если x_N — висячая вершина, то просто удаляем ее. Если x_N имеет только одного потомка, соединяем предка вершины x_N ребром с потомком вершины x_N и, таким образом, удаляем вершину x_N . Если же вершина x_N имеет двух потомков, то проделываем описанные операции с одним из ее потомков, заботясь

о соответствующих изменениях величин $\bar{S}(x)$. Если же, проделав все преобразования, мы выяснили, что удалять вершину нет смысла, то исправляем величины $\bar{S}(x)$, но не реконструируем дерево.

3.6. Вывороченные деревья. Под вывороченными деревьями мы будем понимать бинарные деревья, у которых все висячие вершины расположены на одном уровне. К вывороченным деревьям относятся такие классы деревьев, как H -деревья, или деревья соседства, HV -деревья, или братские деревья, и др.

3.7. Бинарное дерево называется H -деревом, если все висячие вершины находятся на одном и том же уровне и каждая вершина с единственным потомком имеет правого соседа с двумя потомками.

Теорема 16. *H -дерево высоты h имеет по крайней мере $(3/2)^h$ висячих вершин.*

Доказательство проводится индукцией по h . Очевидно, что теорема справедлива для $h=0$ и для $h=1$. Каждое H -дерево T' высоты $h+1$ состоит из H -дерева T высоты h , имеющего по индуктивному предположению $n \geq (3/2)^h$ висячих вершин. Каждой висячей вершине такого дерева сопоставлены один или два потомка, причем каждой второй вершине нужно сопоставлять двух потомков. Для четного n дерево T' имеет самое меньшее

$$n/2 + 2n/2 = (3/2)n \geq (3/2)^{h+1}$$

висячих вершин, для нечетного n — по меньшей мере

$$(n-1)/2 + 2(n-1)/2 + 2 = 1/2 + (3/2)n > (3/2)^{h+1}$$

висячих вершин.

Пример H -дерева приведен на рис. 4.32.

Пусть H -дерево имеет по крайней мере две висячие вершины. Добавление новой вершины может быть произведено лишь на самом нижнем уровне, т. е. добавлена может быть только висячая вершина. При добавлении новой вершины у некоторой вершины может оказаться

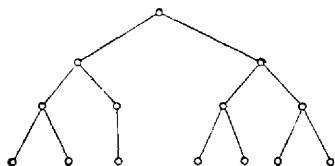


Рис. 4.32.

три потомка, а при удалении может появиться вершина без потомков на уровне выше уровня висячих вершин. В обоих случаях восстановление структуры

И-дерева происходит с помощью алгоритма упрощения.

Алгоритм упрощения (в вершине v):

1. Если вершина v не имеет потомков, то удалить v и вызвать алгоритм упрощения для предка вершины v .

2. Если вершина v имеет в точности одного потомка, то возможны четыре случая:

а) Вершина v имеет соседа v' с одним потомком. В этом случае потомка вершины v перевешиваем к v' , удаляем v и применяем алгоритм упрощения к предку вершины v .

б) Вершина v не имеет соседей. В этом случае удаляем вершину v и делаем ее потомка новым корнем.

в) Вершина v имеет левого соседа v' с двумя потомками и не имеет правого соседа. В этом случае перевесим правого потомка вершины v к вершине v' и применяем алгоритм упрощения к вершине v' .

г) Вершина v имеет правого соседа с двумя потомками и либо не имеет левого соседа, либо имеет левого соседа с двумя потомками. В этом случае никакие преобразования не нужны.

3. Если вершина v имеет точно двух потомков, то преобразования не нужны.

4. Если вершина v имеет трех потомков, то возможны следующие два случая:

а) Вершина v имеет соседа v' с одним потомком. В этом случае перевесим одного потомка вершины v вершине v' .

б) Вершина v не имеет соседа с одним потомком. В этом случае образуем новую вершину v' , единственный потомок которой был самым левым потомком вершины v . Если v была корнем дерева, то образуем новый корень с потомками v и v' . Если же вершина v имеет предка, то подвесим вершину v' дополнительным потомком к предку вершины v и применим алгоритм упрощения к предку вершины v .

Алгоритмы включения и исключения имеют трудоемкость $O(\log n)$, где n — число висячих вершин.

3.8. Будем говорить, что вершина \bar{x} есть *брат* вершины x и записывать $\bar{x} = \beta x$, если x и \bar{x} имеют одного и того же предка. Бинарное дерево называется *ИВ-деревом*, если все висячие вершины находятся на одном и том же уровне и если каждая вершина x с одним потомком имеет брата βx с двумя потомками. Деревья,

показанные на рис. 4.33, б и 4.33, в, представляют собой примеры братских деревьев, в отличие от дерева на рис. 4.33, а.

Сравнение *НВ*-деревьев с *Н*-деревьями показывает, что существуют *Н*-деревья, не являющиеся *НВ*-деревьями, и, наоборот, существуют *НВ*-деревья, не являющимися *Н*-деревьями.

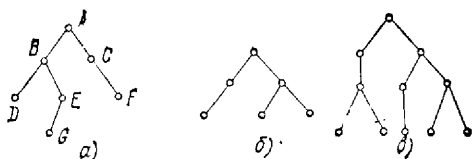


Рис. 4.33.

Теорема 17. *НВ-дерево высоты h имеет по крайней мере $\text{Fib}(h+2)$ висячих вершин. Здесь $\text{Fib}(h)$ обозначает h -е число Фибоначчи: $\text{Fib}(1) = \text{Fib}(2) = 1$, $\text{Fib}(h+2) = \text{Fib}(h) + \text{Fib}(h+1)$.*

Доказательство. *НВ*-дерево с наименьшим числом висячих вершин при заданной высоте получают, делая на каждом уровне число вершин с одним потомком как можно больше, а число вершин с двумя потомками как можно меньше. Обозначим через $N_1(h)$ наибольшее число вершин на уровне h с одним потомком, а через $N_2(h)$ — наименьшее число вершин с двумя потомками на уровне h . Тогда число вершин на уровне h в *НВ*-дереве с возможно большим числом висячих вершин равно $N(h) = N_1(h) + N_2(h)$. Из определения *НВ*-дерева следует, что если вершина на уровне h имеет одного потомка, то у нее на уровне $h+1$ должен быть потомок с двумя потомками (или без потомков, т. е. висячая вершина). Если вершина на уровне $h+1$ имеет двух потомков, то один из ее потомков на уровне $h+1$ обязательно имеет только одного потомка. Отсюда следует:

$$N_1(h+1) = N_2(h), \quad N_2(h+1) = N_1(h) + N_2(h).$$

Так как корень *НВ*-дерева должен всегда иметь двух потомков, то $N_1(0) = 0$, $N_2(0) = 1$. Из этих соотношений следует, что $N_2(1) = N_1(0) + N_2(0) = 1$ и $N_2(h+2) = N_1(h+1) + N_2(h+1) = N_2(h) + N_2(h+1)$. Кроме того, для всех $h \geq 0$ имеем $N_2(h) = \text{Fib}(h+1)$. Отсюда вы-

текает, что

$$N(0) = N_1(0) + N_2(0) = 1 = \text{Fib}(2),$$

$$\begin{aligned} N(h+1) &= N_1(h+1) + N_2(h+1) = N_2(h) + N_2(h+1) = \\ &= \text{Fib}(h+1) + \text{Fib}(h+2) = \text{Fib}(h+3). \end{aligned}$$

Итак, для любого $h \geq 0$ справедливо соотношение $N(h) = \text{Fib}(h+2)$. Теорема доказана.

Поскольку $\text{Fib}(n)$ есть целое число, следующее за числом $(\alpha^n - \beta^n)/\sqrt{5}$, где $\alpha = (1 + \sqrt{5})/2$, $\beta = (1 - \sqrt{5})/2$, то наименьшее число висячих вершин в HB -дереве высоты h менее чем на единицу отличается от числа $(\alpha \cdot 2/\sqrt{5})\alpha^h = 1,170 \cdot 1,618^h$.

Включение новой вершины, так же как и в случае H -деревьев, возможно лишь на уровне висячих вершин. При этом возможно появление вершины с тремя потомками. Восстановление структуры HB -деревя производится с помощью алгоритма упрощения, устроенного так, что его применение ограничено вершинами с тремя потомками, причем либо все три потомка — висячие вершины, либо по меньшей мере два из них сами имеют в точности по два потомка.

Алгоритм упрощения (в вершине v):

1. Если вершина v имеет в точности два потомка, то никакие преобразования не нужны.

2. Если вершина v имеет в точности трех потомков, то возможны два случая:

а) Вершина v не имеет братьев. Создадим вершине v брата v' и передадим ему в качестве потомка одного из потомков вершины v , имеющего, в свою очередь, двух потомков. Если вершина v была корнем, то образуем новый корень — предка вершин v и v' .

б) Вершина v имеет брата w . Тогда, если w имеет точно одного потомка, то передадим ему одного подходящего потомка вершины v . Если же брат w имеет точно двух потомков, то создадим для v дополнительного (третьего) брата v' и передадим ему в качестве потомка одного из потомков вершины v , имеющего по крайней мере двух потомков. Затем применяем алгоритм упрощения к предку вершины v .

Нетрудно увидеть, что в случае б), когда вершина v имеет в точности одного потомка, этот потомок должен быть либо висячей вершиной, либо вершиной

с двумя потомками. Вновь введенный потомок вершины w имеет в каждом случае брата с двумя потомками (если он не является висячей вершиной).

Удаление вершины есть в любом случае удаление висячей вершины, поэтому единственным осложнением может быть лишь появление внутренней вершины без потомков, т. е. висячей вершины на уровне выше уровня остальных висячих вершин. В этом случае применяется другой алгоритм упрощения.

Алгоритм упрощения при удалении (в вершине v):

1. Если вершина v не имеет потомков, то удаляем ее и применяем алгоритм упрощения к предку вершины v .

2. Если вершина v имеет точно одного потомка, то либо вершина v не имеет братьев, либо имеет хотя бы одного брата.

а) Вершина v не имеет братьев. Если при этом вершина v не имеет предка, т. е. является корнем, то удаляем вершину v и делаем единственного потомка вершины v новым корнем. В противном случае предок вершины v имеет брата — вершину w с двумя потомками w' и w'' . Пусть w' будет ближайшим соседом v . Тогда, если w'' имеет двух потомков, то делаем вершину w' братом v и применяем алгоритм упрощения снова к вершине v . Если же вершина w'' имеет одного потомка, то тогда вершина w' должна иметь двух потомков; дадим вершине w'' в качестве дополнительного потомка потомка вершины w' и сделаем вершину w' братом v . После этого применим алгоритм упрощения снова к вершине v .

б) Если вершина v имеет брата w , то преобразования нужны только в том случае, когда вершина w имеет только одного потомка. Мы сделаем потомка вершины w потомком вершины v , удалим вершину w и применим алгоритм упрощения к предку вершины v .

Трудоемкость всех описанных алгоритмов равна $O(\log n)$, где n — число висячих вершин.

3.9. Бинарное дерево называется *HS-деревом*, если справедливо следующее: все висячие вершины находятся на одном уровне, и если вершина x имеет только одного потомка, то этот потомок есть либо висячая вершина, либо сам имеет двух потомков.

Очевидно, что каждое *HB-дерево* является также *HS-деревом*, но обратное неверно. Алгоритм включе-

ния нового слова в *HS*-дерево тот же, что и для *HB*-деревьев; алгоритм удаления несколько отличается в деталях.

3.10. Бинарные деревья малой высоты. AVL-деревья, балансированные по весу деревья, *H*-деревья и *HB*-деревья могут рассматриваться как деревья с малой высотой, поскольку их высоты равны $1,44 \log(n+2)$, $1,071 \log(n+1)$, $1,709 \log n$, $1,44 \log(n+1)$ соответственно. Но зачастую бывают нужны деревья как можно меньшей высоты. Докажем существование для любого $\varepsilon > 0$ балансированных выровненных бинарных деревьев с высотой $(1+\varepsilon) \log n + 1$.

Напомним, что вершина v_i называется непосредственно правым (непосредственно левым) соседом вершины v_0 , если v_0 и v_i — вершины одного и того же уровня и вершина v_i появляется непосредственно справа (слева) от вершины v_0 в графическом изображении дерева, содержащего v_0 и v_i в качестве вершин.

Для любого положительного целого k определим класс k -деревьев. Упорядоченное бинарное дерево называется k -деревом, если:

а) все висячие вершины располагаются на одном и том же уровне;

б) вершина v с одним потомком имеет по крайней мере одного правого соседа и первые k правых соседей (или все правые соседи, если их число меньше k) вершины v имеют двух потомков.

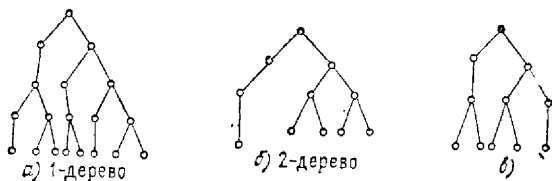


Рис. 4.34.

Замечание. Требование иметь хотя бы одного соседа гарантирует отсутствие вырожденных деревьев с единственной висячей вершиной.

Любое k -дерево есть k' -дерево для $k' \leq k$. 1-деревья — это в точности *H*-деревья. На рис. 4.34 изображены 1-дерево, 2-дерево и дерево, которое не является k -деревом ни для какого k .

Теорема 18. k -дерево высоты h имеет по крайней мере $N_k(h)$ висячих вершин, где

$$N_k(h) = \begin{cases} 1, & \text{если } h = 0; \\ 2^{h-1} + 1, & \text{если } h \geq 1 \text{ и } 2^{h-2} + 1 < k + 3; \\ (2 - 1/(k+1))^{h-1} & \text{в остальных случаях.} \end{cases}$$

Доказательство проведем индукцией по h . Ясно, что теорема справедлива для $h = 0$ и $h = 1$. Если не существует по крайней мере $k + 3$ вершин на уровне $h \geq 2$, то самое большее одна вершина на уровне h имеет только одного потомка. Пусть h_0 — наименьшее число, такое, что $2^{h_0-1} + 1 \geq k + 3$. (Отсюда, кстати, следует, что $h_0 \geq 3$ для каждого $k \geq 1$.) Следовательно, для каждого $h \in [2, h_0]$ имеет место соотношение $N_k(h) \geq 2^{h-1} + 1$.

Теперь докажем, что для всех $h > h_0$ справедливо неравенство $N_k(h) \geq (2 - 1/(k+1))^h + 1$. Если $h = h_0 + 1$, то из минимальности h_0 имеем $2^{h_0-2} + 1 < k + 3$. Следовательно, $2^{h_0-1} + 1 \leq 2(k+1) + 1$. Для $N_k(h_0)$ вершин на уровне h_0 имеем

$$k + 3 \leq 2^{h_0-1} + 1 = N_k(h_0) \leq 2(k+1) + 1 = 2k + 3.$$

Поэтому самое большее две вершины на уровне h_0 имеют единственного потомка, откуда следует, что минимальное число вершин на уровне $h_0 + 1$ удовлетворяет неравенству

$$\begin{aligned} N_k(h_0 + 1) &\geq 2N_k(h_0) - 2 = 2(2^{h_0-1} + 1) - 2 = 2^{h_0} = \\ &= 2^{h-1} \geq (2 - 1/(k+1))^{h-1}. \end{aligned}$$

Рассмотрим теперь шаг индукции перехода от h к $h + 1$.

$$\begin{aligned} N_k(h+1) &\geq (\text{наибольшее число вершин на уровне } h \\ &\text{с единственным потомком}) + 2(\text{наименьшее число вершин на уровне } h \\ &\text{с двумя потомками}) \geq \left(\frac{N_k(h) - 2}{k+1} \right) + \\ &+ 2 \left(k \frac{N_k(h) - 2}{k+1} + 1 \right) = [N_k(h) - 2] \left(2 - \frac{1}{k+1} \right) + 3 \geq \\ &\geq \left(\left(2 - \frac{1}{k+1} \right)^{h-1} - 2 \right) \left(2 - \frac{1}{k+1} \right) + 3 = \left(2 - \frac{1}{k+1} \right)^h - \\ &- 4 + \frac{2}{k+1} + 3 \geq \left(2 - \frac{1}{k+1} \right)^h \end{aligned}$$

и т. д. Включение новой вершины происходит с помощью алгоритма k -включения.

Алгоритм:

1. Если вершина v после включения новой вершины имеет точно двух потомков, то никакие преобразования не нужны.

2. Пусть v имеет трех потомков, тогда:

а) Хотя бы одна вершина из k левых или правых соседей вершины v имеет точно одного потомка. Берем одного из ближайших соседей с единственным потомком и вызываем процедуру ПЕРЕБРОС(v', v).

б) Если ни одна вершина из k левых или правых соседей не имеет точно одного потомка, то создадим новую вершину v' с левым потомком вершины v в качестве ее единственного потомка. Возможны два подслучая: вершина v имеет или не имеет предка. Если не имеет, то создадим новую вершину w в качестве нового корня с потомками v и v' . В противном случае подвесим v' в качестве нового потомка к предку вершины v и применим алгоритм k -включения к вершине — предку v .

Процедура ПЕРЕБРОС(v_1, v_2), вызываемая в случае 2а), определена для двух вершин v_1 и v_2 , $v_1 \neq v_2$, расположенных на одном и том же уровне. Вызов этой процедуры предполагает, что все вершины между v_1 и v_2 на том же уровне, что и v_1 , имеют двух потомков, v_1 имеет только одного потомка, вершина v_2 — трех.

процедура ПЕРЕБРОС(v_1, v_2);

1. начало

2. если v_2 — правый сосед вершины v_1 то

3. $w :=$ непосредственный правый сосед вершины v_1 ;

4. иначе $w :=$ непосредственный левый сосед вершины v_1 ;

5. $r :=$ потомок вершины w , являющийся непосредственным соседом потомка вершины v_1 ;

6. сделать r дополнительным потомком вершины v_1 ;

7. если $w \neq v_2$ то ПЕРЕБРОС(w, v_2);

8. иначе стоп;

конец

Удаление вершины есть удаление всячей вершины. Для восстановления структуры k -дерева используется алгоритм k -удаления (в вершине v).

Алгоритм:

1. Если вершина v не имеет потомков, то удаляем ее и применяем алгоритм k -удаления к предку вершины v .

2. Если вершина v имеет одного потомка, то возможны два случая:

2.1. Одна вершина из k левых или правых соседей вершины v имеет одного потомка. Находим вершину v' с единственным потомком, который является ближайшим соседом такого вида. Вызываем процедуру ПЕРЕБРОС(v' , v), удаляем v и применяем алгоритм k -удаления к предку вершины v .

2.2. Ни один из k левых или правых соседей не имеет в точности одного потомка. Здесь возможен целый ряд вариантов:

2.2.1. Вершина v имеет по крайней мере одного правого соседа; преобразования не нужны.

2.2.2. Вершина v не имеет правого соседа. Здесь также возможен ряд вариантов:

2.2.2.1. Если вершина v не имеет левого соседа, то удаляем v и берем единственного потомка вершины v в качестве нового корня.

2.2.2.2. Вершина v имеет по крайней мере одного правого и самое большее $k \geq 1$ левых соседей. Пусть v' — непосредственный левый сосед вершины v . Сделаем правого потомка вершины v' левым потомком вершины v , так что у v теперь два потомка.

2.2.2.3. Вершина v имеет по крайней мере $k+1$ левых соседей. Пусть w есть $k+1$ -й левый сосед вершины v . Имеют место две возможности:

а) Вершина w имеет только одного потомка. В этом случае вызываем процедуру ПЕРЕБРОС(w , v), удаляем вершину v и применяем алгоритм k -удаления к предку вершины v .

б) Вершина w имеет двух потомков. Пусть v' — непосредственный левый сосед вершины v . Сделаем правого потомка вершины v' левым потомком вершины v .

Трудоемкость алгоритмов включения и удаления вершин для k -дерева высоты h пропорциональна его высоте h .

Теорема 19. Для каждого $\varepsilon > 0$ существует такое k , что класс k -деревьев есть класс сбалансированных бинарных деревьев высоты $\leq (1 + \varepsilon) \log n + 1$, где n — число висячих вершин.

Доказательство. Из описания алгоритмов видно, что класс k -деревьев есть класс балансированных бинарных деревьев для каждого k . Остается показать, что можно выбрать k подходящим образом. Выберем $k \geq \geq 1/(2 - 2^{1/(1+\varepsilon)}) - 1$. По теореме 18 для k -деревьев высоты h с n висячими вершинами и $k + 1 \geq 1/(2 - 2^{1/(1+\varepsilon)})$ имеем $(2 - 1/(k + 1))^{h-1} \leq n$, т. е. $h - 1 \leq \leq \log n / \log (2 - 1/(k + 1)) \leq \log n / \log 2^{1/(1+\varepsilon)} = (1 + \varepsilon) \log n$. Теорема доказана.

В заключение приведем высоты k -деревьев для малых значений k . Заметим, что для $k \geq 2$ высоты меньше, чем высоты любого известного ранее класса балансированных бинарных деревьев (табл. 4.4).

Т а б л и ц а 4.4

k	(Высота k -дерева) — 1	k	(Высота k -дерева) — 1
1	$1,71 \log n$	5	$1,14 \log n$
2	$1,36 \log n$	9	$1,08 \log n$
3	$1,24 \log n$	99	$1,01 \log n$
4	$1,18 \log n$		

3.11. Выровненное дерево называется m -арным выровненным деревом, если оно удовлетворяет следующим условиям:

- а) каждая вершина имеет не более m потомков;
- б) каждая вершина, кроме корня и висячих вершин, имеет не менее $m/2$ потомков;
- в) корень, если он не является висячей вершиной, имеет не менее двух потомков.

При добавлении новой висячей вершины может появиться вершина с $m + 1$ потомками. В этом случае она расщепляется на две вершины, у каждой из которых либо по $((m + 1)/2)$ потомков, если m нечетно, либо у одной $\lfloor (m + 1)/2 \rfloor$ потомков, а у второй $\lfloor (m + 1)/2 \rfloor + 1$. Если у предка расщепленной вершины менее $((m + 1)/2)$ потомков, то процесс включения новой вершины заканчивается; в противном случае процедура расщепления продолжается. Если расщепляемая вершина есть корень, то образуется новый корень, потомками которого становятся новые вершины.

Пример включения новой вершины в 5-арное дерево показан на рис. 4.35.

Удаление вершины происходит также на уровне висячих вершин. Если при удалении вершины у ее предка остается менее $m/2$ потомков, то вершина-предок сливается с одним из своих братьев (либо с ближайшим справа, либо с ближайшим слева). Если получившаяся после слияния вершина имеет более $m + 1$ потомков, то она расщепляется, как это делается при

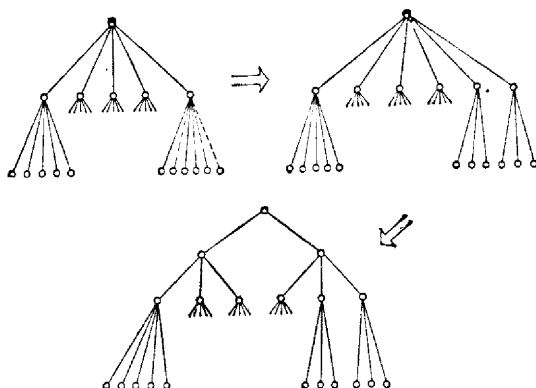


Рис. 4.35.

включении новой вершины. Так как при слиянии вершин может возникнуть вершина с малым числом потомков на более высоком уровне, повторяется процедура слияния, и т. д.

Пример удаления вершины из 5-арного дерева показан на рис. 4.36.

3.12. Плотные m -арные деревья. Вершина v с единственным потомком sv называется *унарной*, вершина с двумя потомками — *бинарной*; вершина, имеющая наибольшее возможное число потомков m или являющаяся висячей, называется *насыщенной*; в противном случае о ней говорят как о *ненасыщенной*.

m -арное дерево T называется r -плотным, где r — натуральное число, $r \in [1 : m - 1]$, если выполнены следующие условия:

- а) корень дерева T по крайней мере бинарен;
- б) каждая ненасыщенная вершина, отличная от корня, имеет не менее r насыщенных братьев;
- в) все висячие вершины располагаются на одном уровне.

Сформулируем некоторые следствия из данного определения.

Следствие 1. Если вершина v есть единственный потомок своего предка — вершины pu , то она должна быть насыщенной.

Следствие 2. Если вершина v имеет k потомков, $k \in [2 : t]$, то существует самое большее одна ненасыщенная вершина среди этих потомков при $k \leq r$ и не менее r насыщенных потомков при $k \geq r + 1$.

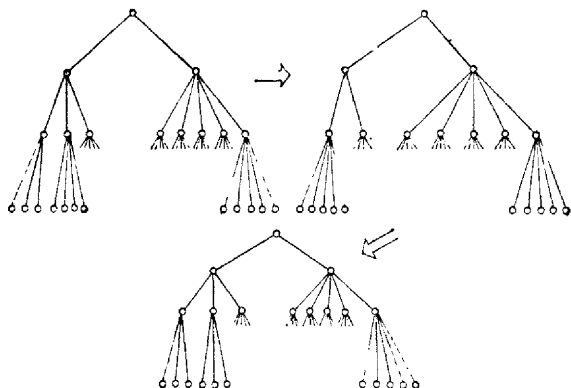


Рис. 4.36.

Класс t -арных деревьев называется *плотным*, если он представляет собой класс r -плотных t -арных деревьев для некоторого r .

Мы будем говорить о *слабо плотных* t -арных деревьях, если имеем в виду класс 1-плотных деревьев, и о *сильно плотных* t -арных деревьях, если имеем в виду класс $t - 1$ -плотных деревьев. Заметим, что существует только один класс плотных бинарных деревьев и он совпадает с классом братских деревьев, или *НВ*-деревьев. Существуют два класса плотных 3-арных, или тернарных деревьев. На рис. 4.37 показаны два тернарных дерева; первое представляет собой пример сильно плотного, а второе — слабо плотного тернарного дерева. Оба имеют по 11 висячих вершин.

Заметим, что для данной высоты h полное t -арное дерево, т. е. дерево, в котором каждая внутренняя вершина имеет t потомков, имеет наибольшее число висячих вершин среди всех r -плотных t -арных деревьев высоты h .

Пусть T_h есть r -плотное t -арное дерево высоты h с наименьшим возможным числом висячих вершин и с

насыщенным корнем. Тогда дерево T_{h+2} получается (с точностью до симметрии) из $m-r$ копий дерева T_h и r копий дерева T_{h+1} , как показано на рис. 4.38. Кроме того, T_0 есть дерево, состоящее из одной вершины, а T_1 имеет вид, показанный на рис. 4.39.

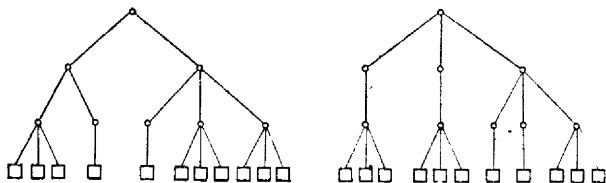


Рис. 4.37.

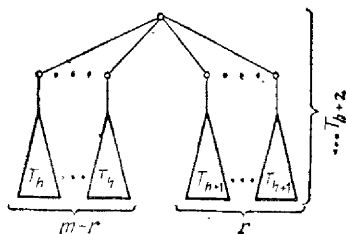


Рис. 4.38.

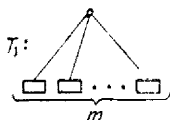


Рис. 4.39.

Это дает следующее рекуррентное соотношение для числа $l_{\min}(r, m, h)$ висячих вершин r -плотного m -арного дерева T_h :

$$\begin{aligned} l_{\min}(r, m, h+2) &= \\ &= (m-r)l_{\min}(r, m, h) + rl_{\min}(r, m, h+1), \\ l_{\min}(r, m, 0) &= 1, \quad l_{\min}(r, m, 1) = m. \end{aligned} \quad (4.34)$$

Так как корень r -плотного m -арного дерева должен быть по крайней мере бинарным, то r -плотное m -арное дерево высоты h с наименьшим возможным числом висячих вершин должно иметь вид, показанный на рис. 4.40. Поэтому r -плотное m -арное дерево высоты $h \geq 1$ имеет по крайней мере $L_{\min}(r, m, h)$ висячих вершин, где

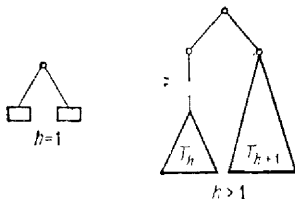


Рис. 4.40.

$$\begin{aligned} L_{\min}(r, m, h) &= \\ &= \begin{cases} 2, & \text{если } h = 1; \\ l_{\min}(r, m, h-1) + l_{\min}(r, m, h-2), & \text{если } h > 1. \end{cases} \end{aligned}$$

Для того чтобы решить соотношение (4.34), заметим вначале, что $l_{\min}(r, m, h) = a_{h+1}$, где $a_n = (m-r)a_{n-2} + ra_{n-1}$, $n \geq 2$, $a_0 = 1$, $a_1 = 1$. Используя известные методы решения рекуррентных соотношений, получим решение соотношения (4.34) в виде

$$a_n = A_1 \alpha_1^n + A_2 \alpha_2^n,$$

где $\alpha_1 = (1/2)(r + \sqrt{4(m-r) + r^2})$, $\alpha_2 = (1/2)(r - \sqrt{4(m-r) + r^2})$, $A_1 = (1 - \alpha_2)/(\alpha_1 - \alpha_2)$, $A_2 = (\alpha_1 - 1)/(\alpha_1 - \alpha_2)$.

Для всех $h \geq 1$ имеем

$$\begin{aligned} L_{\min}(r, m, h) &= a_h + a_{h-1} = \\ &= (A_1 \cdot A_1 / \alpha_1) \alpha_1^h + (A_2 \cdot A_2 / \alpha_2) \alpha_2^h. \end{aligned}$$

Отсюда вытекает, что для специального случая бинарных деревьев ($m = 2$, $r = 1$)

$$\alpha_1 = \frac{1 + \sqrt{5}}{2}, \quad \alpha_2 = \frac{1 - \sqrt{5}}{2},$$

$$A_1 = \frac{1}{\sqrt{5}} \cdot \frac{1 + \sqrt{5}}{2}, \quad A_2 = -\frac{1}{\sqrt{5}} \cdot \frac{1 - \sqrt{5}}{2},$$

п, следовательно, минимальное число висячих вершин в братских деревьях высоты h равно приблизительно $1,171 \cdot (1,618)^h$.

Далее, слабо плотное тернарное дерево ($m = 3$, $r = 1$) высоты h имеет не менее 2^h висячих вершин, а наименьшее число висячих вершин сильно плотного тернарного дерева ($m = 3$, $r = 2$) равно

$$L_{\min}(2, 3, h) = \frac{\sqrt{2}}{2} ((1 + \sqrt{2})^h - (1 - \sqrt{2})^h).$$

Заметим, что число висячих вершин сильно плотного m -арного дерева высоты h пропорционально $(m-1)^h$. В этом случае $r = m-1$ и, следовательно, $|\alpha_2| < 1$, а α_1 приблизительно равно $m-1$ (для больших m).

Библиографический комментарий

Первой работой, посвященной оптимальным нумерациям ордерова — укладкам, является заметка А. П. Ершова [28], изданная в 1958 г. Его метод был переоткрыт другими авторами, изложен на языке теории графов Р. Реджеёвским и стал широко известен благодаря переводу работы последнего на русский язык [71]. Доказательства теорем 1 и 2 могут быть найдены в этой работе (см. также [25]). Оптимальная укладка взвешен-

ного дерева рассматривалась в работе [79], развитие в сторону введения регистровой памяти для организации вычислений было сделано Р. Сети и Дж. Ульманом. Подробнее об этом см. в главе 6. Алгоритм построения оптимальной по длине укладки ордерера излагается по работе М. А. Шейдвассера [85], алгоритм укладки неориентированного дерева — по работе М. А. Иорданского [36] (см. также [34]). Вывод оценок для длины и ширины ордерера можно найти в [86]. Точное решение задачи об укладке неориентированных деревьев достаточно сложно. Один из первых результатов относится к 1971 г., когда в работе [64] был предложен метод ветвей и границ, затем в 1973 г. был применен метод последовательного анализа вариантов с экспоненциальной трудоемкостью [16]. Точное решение с трудоемкостью $O(n^3)$ было дано в 1976 г. в работе [21]. Пока неизвестно, существует ли точный алгоритм с меньшей трудоемкостью. Что касается приближенных методов, то в работе [86] с помощью минимальных укладок корневых ориентированных деревьев удалось получить приближение к оптимальному решению не хуже чем в два раза. Трудоемкость соответствующего алгоритма не более $O(n^2 \log n)$. Поскольку оптимальные нумерации таких деревьев являются плоскими, то, как показано в [18], эффективность поиска приближенного решения можно увеличить в $\log n$ раз, используя предложенный в [17] алгоритм построения минимальной плоской укладки деревьев. Наконец, в [36] был предложен алгоритм построения плоских минимальных укладок с трудоемкостью $O(n)$, и там же было показано, что минимальная длина укладки в классе плоских укладок превосходит соответствующую величину в классе всех укладок не более чем в полтора раза (предварительное сообщение об этом алгоритме было дано в [35]). Построение оптимальных по числу соседней связности укладок деревьев и бесконтурных орграфов излагается по работе [50].

Алгоритм K построения оптимальных бинарных деревьев был разработан Д. Кнудом в 1971 г. и позднее вошел в его книгу [48], в соответствии с которой и излагается здесь. Описанный в п. 2.3 алгоритм построения почти оптимальных деревьев принадлежит Б. Бруно и Э. Коффману [105], top-down-алгоритм — Е. Нисикаве, Е. Еспиде и Т. Фукумуре [140]. Приводимый в п. 2.4 анализ принадлежит Ю. Нивергельту и К. Вонгу [139], оценки для балансированных деревьев принадлежат К. Мельхорну [134] и П. Байеру [99], пример того, что правило 1 строит «плохие» деревья, принадлежит К. Мельхорну. Первая эвристика построения динамических деревьев принадлежит Дж. Баеру [95], вторая, приводящая к классу деревьев $B_{1/3}$, — К. Унтерауэру [159]; упоминаемые на шаге 5 алгоритма вспомогательные действия по восстановлению структуры дерева из класса $B_{1/3}$ представляют собой алгоритм М. Фредмана [143]. Что касается остальных помещенных в § 2 алгоритмов и результатов, то за ссылками на литературу мы отсылаем к обзору автора [26]. Построение оптимальных каркасов, требующее полного перебора, дано в соответствии с работами [58, 59, 73].

За ссылками на литературу к § 3 мы также отсылаем к обзору [26], указав только на доступные работы по отдельным классам деревьев. Это работы [2, 48, 55] (АВЛ-деревья), [7] (2-3-деревья), [48, 72] (В-деревья), [72] (ВВ-деревья).

ГЛАВА 5

РАЗРЕЗАНИЯ И РАСКРАСКА ГРАФОВ

§ 1. Разрезания графов

1.1. Постановка задачи. Пусть задан взвешенный граф $G = (X, U)$, $|X| = n$, вершины v_i и дуги u_{ij} которого имеют веса c_i и c_{ij} соответственно, и пусть задан набор чисел $\{m_k\}$, $m_k > 0$, $k \in [1:r]$. Требуется найти разрезание $R(X) = (X_1, \dots, X_k)$ графа на k кластеров — подграфов $\langle X_k \rangle$, для которых выполнены следующие условия:

$$\text{а) } X = \bigcup_{k=1}^r X_k, \quad X_i \cap X_j = \emptyset, \quad i \neq j; \quad (5.1)$$

$$\text{б) } C^{(k)} = \sum_{v_i \in X_k} c_i \leq m_k, \quad k = 1, \dots, r; \quad (5.2)$$

$$\text{в) } \sum_{k=1}^r \sum_{v_i, v_j \in X_k} c_{ij} \rightarrow \max \quad (5.3)$$

или

$$\text{г) } \sum_{k=1}^{r-1} \sum_{l=k+1}^r \sum_{v_i \in X_k, v_j \in X_l} c_{ij} \rightarrow \min. \quad (5.4)$$

Ограничение (5.2) может иметь вид

$$C^{(k)} = M, \quad (5.2a)$$

$$C^{(k)} \leq M, \quad (5.2б)$$

$$C^{(k)} = m_k, \quad (5.2в)$$

$$C^{(k)} \in [A, B]. \quad (5.2г)$$

Задача о разрезании во многом аналогична известной в исследовании операций одномерной задаче о рюкзаке. Пусть рюкзак альпиниста вмещает M килограммов груза, и пусть имеется набор различных пред-

метов, которые он хотел бы взять с собой. Каждый предмет, кроме веса, характеризуется еще некоторой ценностью, а сумма весов всех предметов превышает M . Математическая постановка задачи следующая:

Пусть c_t — вес предмета $t \in [1:n]$, w_t — ценность предмета t , M — емкость рюкзака. Максимизировать $\sum_{t=1}^n w_t \gamma_t$ при условии $\sum_{t=1}^n c_t \gamma_t \leq M$, где

$$\gamma_t = \begin{cases} 1, & \text{если предмет } t \text{ находится в рюкзаке;} \\ 0, & \text{в противном случае,} \end{cases}$$

Задача о разрезании графов может рассматриваться как обобщение этой задачи, а именно как задача о размещении связанных между собой предметов с заданными весами по нескольким рюкзакам или кластерам, каждый из которых имеет емкость M .

Задача о разрезании графа (бесконтурного ориентированного) на минимальное число частей сводится к задаче целочисленного программирования. Пусть n — число вершин в графе, q — число кластеров, полученных после разрезания (заметим, что значение q до решения задачи неизвестно). Пусть, далее, m_{ij} — число дуг, ведущих из кластера X_i в кластер X_j и наоборот; M — заданная константа. Введем переменные γ_{ij} :

$$\gamma_{ij} = \begin{cases} 1, & \text{если вершина } v_i \text{ включена в кластер } X_j; \\ 0 & \text{в противном случае.} \end{cases}$$

Задача оптимального разрезания состоит в нахождении двоичных γ_{ij} , таких, что выполняются ограничения

$$\sum_{i=1}^n \gamma_{ij} \leq M, \quad j=1, 2, \dots, q, \quad \sum_{j=1}^q \gamma_{ij} = 1, \quad i=1, \dots, n, \quad (5.5)$$

и таких, что функционал

$$J = \sum_{i=1}^n \sum_{h=1}^n m_{ih} \sum_{j=1}^q \gamma_{ij} \gamma_{hj} \quad (5.6)$$

достигает своего минимума. Ограничения (5.5), так же как и (5.2), могут иметь различный вид.

1.2. Разрезом $R(X_1, X_2)$ в неориентированном графе, разделяющим вершины a и b , называется множество ребер вида (x, y) , где $x \in X_1$, $y \in X_2$, удаление которых разбивает граф по крайней мере на две компоненты

связности G_a и G_b , причем $a \in G_a$, $b \in G_b$. Разрез называется *собственным*, если ни одно его собственное подмножество разрезом не является.

Разрезом $R(X_a, X_b)$ в ориентированном графе $G(X, U)$, разделяющим вершины a и b , называется множество дуг вида (x, y) , где $x \in X_a$, $y \in X_b$, такое, что любой путь из $a \in X_a$ в $b \in X_b$ содержит по крайней мере одну дугу из $R(X_a, X_b)$.

Пусть дан орграф со взвешенными дугами, и пусть c_{ij} — вес дуги u_{ij} . *Величиной* разреза $R(X_1, X_2)$ называется сумма весов входящих в него дуг. *Минимальным разрезом*, разделяющим вершины a и b , называется разрез минимального веса, разделяющий эти вершины. Задача отыскания для заданных вершин a и b минимального разреза, разделяющего их, сводится к отысканию максимального потока по сети с входом a , выходом b и весами c_{ij} , взятыми в качестве пропускных способностей дуг. Положив все c_{ij} равными 1, мы можем найти разрез с наименьшим возможным числом дуг. Для неориентированных графов задача решается аналогично.

З а м е ч а н и е. Ранее мы ввели понятие минимального потока в сети с ограничениями на величину потока снизу. Нетрудно внести соответствующие изменения в определение, чтобы получить определение максимального потока, т. е. потока с наибольшей возможной величиной, удовлетворяющего ограничениям на величину потока по дуге сверху.

Выделение блоков в графе и нахождение минимального разреза представляют задачу разрезания графа в ее «естественном виде», т. е. когда кластеры определяются структурой графа и ничем более.

1.3. Разрезание деревьев методом динамического программирования. Алгоритм предназначен для решения задачи (5.1), (5.2б), (5.4), т. е. задачи:

Найти разбиение вершин дерева $T(X, U)$, каждая из которых имеет вес c_i , на k кластеров X_1, \dots, X_k таких, что

$$X = \bigcup_{i=1}^k X_i, \quad X_i \cap X_j = \emptyset, \quad i \neq j, \quad (5.1)'$$

$$\sum_{v_i \in X_j} c_i \leq M, \quad (5.2б)'$$

$$\sum_{i=1}^{k-1} \sum_{l=i+1}^k \sum_{v_i \in X_l, v_j \in X_i} c_{ij} \rightarrow \min. \quad (5.4)'$$

Для представления разбиения будем использовать списки вершин кластеров. Каждый кластер, образованный при разбиении дерева T , представляет собой либо поддереву, либо лес, т. е. несвязный граф, каждая компонента которого есть дерево. Однако мы ограничимся такими разбиениями дерева T , у которых каждый кластер представляет собой поддереву. Это ограничение не слишком сильное, так как кластеры оптимального разбиения любого связного графа могут быть модифицированы путем образования из каждого кластера нового множества кластеров, каждый из которых представляет собой связный подграф. Так как исходное разбиение было оптимальным, а новое разбиение имеет вес не больше исходного, такая модификация приводит снова к оптимальному разбиению, в котором каждый кластер соответствует связному графу.

Для удобства обозначений превратим T в ориентированное упорядоченное дерево T' , как это показано на рис. 5.1. При этом вершина v_1 становится корнем (будем считать, что вершина v_i имеет номер i), а потомки каждой вершины, включая корень, упорядочиваются слева направо в порядке возрастания их номеров.

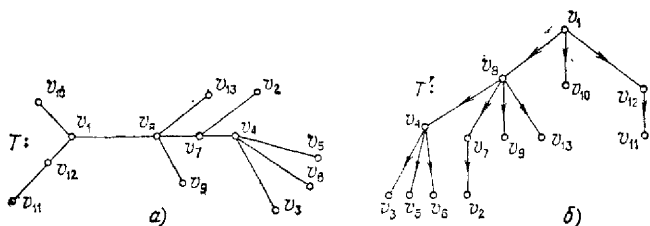


Рис. 5.1.

Динамическое программирование дает возможность использовать основное свойство дерева, а именно его ацикличность, для нахождения глобально оптимального разбиения на основе локальной информации. Алгоритм порождает оптимальное разбиение дерева T' путем нахождения разбиений все увеличивающихся по размеру поддеревьев дерева T' , пока не будет разбито само дерево T' .

Первый шаг в алгоритме разбиения состоит в генерировании тривиального разрезания висячих вершин

дерева T' . Потом формируется множество вершин в дереве T' , у которых все поддеревья уже разрезаны. Предположим, что в этом множестве есть вершина x .

Шаг 1. Найти разрезание поддерева $T_1(x)$, образованного вершиной x и ее первым поддеревом.

Шаг 2. Комбинируем это разрезание и разрезания второго поддерева вершины x для создания разрезания поддерева, образованного вершиной x и ее первыми двумя поддеревьями.

Шаг 3. Комбинируем разрезания, полученные на шаге 2, с разрезаниями третьего поддерева вершины x для создания разрезания поддерева, образованного x и первыми тремя поддеревьями.

Шаг 4. Продолжаем эту процедуру до тех пор, пока не будет разрезано все дерево с корнем в вершине x .

Если в результате разрезания дерева с корнем в x окажется, что предок x становится вершиной, у которой завершено разрезание всех ее поддеревьев, то добавим эту вершину к нашему множеству.

По окончании разрезания дерева с корнем в вершине x удалим x из множества вершин W , обладающих тем свойством, что все их поддеревья разрезаны. Выбираем другую вершину из W и разрезаем соответствующее поддерево. Как только множество W будет исчерпано, алгоритм заканчивает свою работу: оптимальное разрезание получено.

Прежде чем этот алгоритм рассматривать практически, нужно найти некоторый эффективный метод получения разрезаний поддерева, состоящего из вершины w и ее первых $i+1$ деревьев, из имеющегося разрезания поддерева, образованного этой вершиной и ее первыми i поддеревьями, и известного к этому моменту разрезания $i+1$ -го поддерева. Заметим, далее, что число разрезаний поддерева может быть очень велико. Например, рассмотрим дерево, состоящее из вершины, все поддеревья которой суть висячие вершины. Число разрезаний этого дерева, при которых соблюдены ограничения на вес и связность, равно $\sum_{i=0}^{M-1} \binom{n-1}{i}$, где n — число вершин в дереве, а M — ограничение на вес кластера. Для $M > n/2$ эта граница растет, как 2^n . Поэтому при решении задачи нужно удалять большое число разрезаний каждого дерева.

Рассмотрим некоторое поддерево $T_i(w)$ дерева T' на рис. 5.2, состоящее из вершины w и ее первых i поддеревьев. При доказательстве оптимальности мы покажем, что нужно провести максимум M разрезов $T_i(w)$ в процессе работы алгоритма. Если c — вес вершины w , то множество разрезов есть $w^{(i)}$, $i \in [c, M]$,

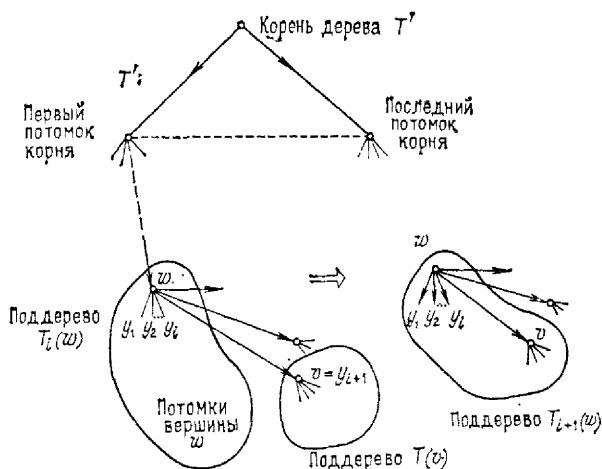


Рис. 5.2.

где каждое разрезание $w^{(i)}$ отличается от других разрезов поддерева $T_i(w)$ следующими свойствами:

1. Разрезание $w^{(i)}$ есть разрезание поддерева $T_i(w)$, для которого вес кластера, содержащего вершину w , равен i .

2. Разрезание $w^{(i)}$ есть разрезание с наибольшим весом среди всех разрезов, обладающих свойством 1.

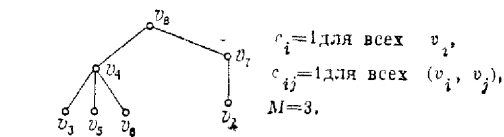
Рассмотрим метод генерирования разрезания поддерева $T_{i+1}(w)$ для данных разрезов поддеревьев $T_i(w)$ и $T(v)$, где вершины поддеревьев $T_i(w)$ и $T(v)$ заключают в себе вершины поддерева $T_{i+1}(w)$. Рассмотрим два поддерева $T_i(w)$ и $T(v)$ на рис. 5.2 и будем считать, что их разрезания уже известны. Чтобы получить разрезание поддерева $T_{i+1}(w)$, получающегося в результате склейки деревьев $T_i(w)$ и $T(v)$, будем использовать две операции:

1. Склеивание кластеров разрезания подграфа $T_i(w)$ с кластерами разрезания подграфа $T(v)$ с помощью слияния вершин в кластерах, содержащих вершины w и v , в один кластер.

2. Склеивание кластеров $w^{(i)}$ и $v^{(j)}$.

Заметим, что любая другая комбинация разрезов $w^{(i)}$ и $v^{(j)}$ нарушает ограничение на связность.

Будем использовать обозначение $C[w^{(i)}, v^{(j)}]$ для разрезания, получающегося в результате применения первой операции. Вес кластера разрезания $C[w^{(i)}, v^{(j)}]$, содержащего вершину w , равен $i + j$, а величина разрезания есть сумма величин $w^{(i)}$ и $v^{(j)}$ и веса дуги (v_i, v_j) . Разрезание, получаемое в результате второй операции, может обозначаться подобным же образом, если мы определим $v^{(0)}$ как разрезание поддерева $T(v)$, которое имеет наибольшую величину среди всех разрезов $T(v)$. (Если вторая операция использовалась для комбинации $w^{(i)}$ с любым разрезанием $v^{(j)}$, результирующее разрезание никогда не будет иметь величину, большую чем $C[w^{(i)}, v^{(0)}]$. Таким образом, мы рассматриваем только комбинации $C[w^{(i)}, v^{(0)}]$ с другими разрезаниями при формировании некоторых $\tilde{w}^{(k)}$, используя вторую операцию.) Тогда $C[w^{(i)}, v^{(0)}]$ изображает разрезание поддерева $T_{i+1}(w)$, у которого кластер, содержащий вершину w , имеет вес i и величина которого равна сумме величин $w^{(i)}$ и $v^{(0)}$.



Пусть

$$s^{(12)} = (v_8, v_4)(v_3)(v_5)(v_6), \quad L(s^{(12)}) = 1,$$

$$7^{(1)} = (v_2)(v_7), \quad L(7^{(1)}) = 0,$$

$$7^{(10)} = (v_2, v_7), \quad L(7^{(0)}) = 1.$$

Тогда

$$C[s^{(12)}, 7^{(1)}] = (v_7, v_8, v_4)(v_2)(v_3)(v_5)(v_6), \quad L(C[s^{(12)}, 7^{(1)}]) = 2,$$

$$C[s^{(12)}, 7^{(0)}] = (v_8, v_4)(v_3)(v_5)(v_6)(v_7, v_2), \quad L(C[s^{(12)}, 7^{(0)}]) = 2.$$

Рис. 5.3.

Заметим, что мы сделали предположение о целочисленности весов всех вершин в дереве T и, следовательно, в дереве T' . Поэтому разрезание $\tilde{w}^{(k)}$ поддерева $T_{i+1}(w)$ выбирается из множества $\{C[w^{(i)}, v^{(k-i)}]\}$,

$C[w^{(2)}, v^{(k-2)}], \dots, C[w^{(k-1)}, v^{(1)}], C[w^{(k)}, v^{(0)}]$ }, где предполагается, что вершины w и v имеют единичные веса (рис. 5.3). Разрезание $C[w^{(a)}, v^{(b)}]$ с наибольшей величиной из этого множества берется в качестве $\tilde{w}^{(k)}$.

Перейдем к описанию алгоритма. Пусть $p(v)$ есть разрезание поддерева $T_i(v)$ с корнем v , содержащего i первых поддеревьев вершины v в качестве своих поддеревьев, $L(p, v)$ — величина разрезания $p(v)$.

А л г о р и т м:

В х о д: Дерево $T = (V, E)$, целочисленные веса $c(v)$ его вершин, веса $c(e)$ дуг, ограничение M на вес кластера.

В ы х о д: Разрезание дерева на k кластеров, вес которых не превышает M .

начало

1. занумеровать вершины дерева T в произвольном порядке;

2. образовать ориентированное упорядоченное дерево T' с корнем в вершине v_1 ;

3. для всех висячих вершин v_i цикл

4. **начало**

5. $p(v_i) := (v_i); i^{(1)} := i^{(0)} := (v_i);$

6. $L(p, v_i) := 0; L(i^{(1)}) := L(i^{(0)}) := 0;$

7. **конец цикла;**

8. для каждой внутренней вершины v_i цикл

9. **начало**

10. $p(v_i) := (v_i); i^{(1)} := (v_i);$

11. $L(p, v_i) := 1; L(i^{(1)}) := 0;$

12. **конец цикла;**

13. сформировать множество W вершин, для поддеревьев которых получены оптимальные разрезания;

примечание на данном шаге в это множество входят лишь те внутренние вершины, у которых потомки суть висячие вершины **конец примечания**

14. для $x_i \in W$ пока W не пусто цикл

15. **начало**

16. для всех $w_k \in \Gamma x_i$ цикл

17. **начало**

18. для j от $c(x_i)$ до M шаг 1 цикл

19. **начало**

20. образовать множество разрезов $\tilde{x}_i^{(j)} = C[x_i, y_j] = \{C[i^{(c)}, k^{(j-c)}], C[i^{(c+1)}, k^{(j-c-1)}], \dots, C[i^{(j)}, k^{(0)}]\}$ с помощью операций, описанных выше;

21. выбрать разрезание $\tilde{x}_i^{(j)}$ из $C[x_a, y_b]$ с наибольшей величиной $L(\tilde{x}_i^{(j)})$;
22. конец цикла;
23. для j от $c(x_i)$ до M шаг 1 цикл
24. начало
25. $i^{(j)} := \tilde{x}_i^{(j)}$;
26. $L(i^{(j)}) := L(\tilde{x}_i^{(j)})$;
27. конец цикла;

примечание обновление разрезаний осуществляется только после завершения всех итераций по j конец примечания

28. $i^{(0)} :=$ разрезание из множества разрезаний с наибольшей величиной;
29. конец цикла;
30. удалить потомков вершины x_i из дерева T' ;
31. удалить вершину x_i из множества W ;
32. если для всех поддеревьев вершины — предка вершины x_i получены оптимальные разрезания то включить предка вершины x_i в множество W ;
33. конец цикла;
34. $i^{(0)}$ есть оптимальное разрезание дерева T' ;
- конец

Покажем, что полученное разрезание оптимально. Рассмотрим некоторое оптимальное разрезание q дерева T' . Предположим, что каждый кластер разрезания q содержит связное поддерево дерева T' . Разобьем все кластеры на три группы:

1. Кластеры, соответствующие поддеревьям, корни которых суть первые i потомков некоторой вершины w .
2. Единственный кластер, который содержит вершину w .
3. Все остальные кластеры.

Приступим к удалению из кластеров разрезания q вершин (исключая вершину w), которые не принадлежат поддереву с корнем, являющимся одним из первых i потомков w . Получившиеся кластеры представляют собой разбиение \hat{w} поддерева $T_i(w)$ на рис. 5.2. Мы теперь утверждаем, что если кластер из \hat{w} , содержащий вершину w , имеет вес j , то величина \hat{w} равна величине разрезания $w^{(j)}$.

Предположим, что величина разрезания \hat{w} меньше, чем величина разрезания $w^{(j)}$. Создадим разрезание p ,

величина которого превышает величину разрезания на то же число, на которое величина разрезания $w^{(j)}$ превышает \hat{w} . Далее:

а) Сделаем кластеры разрезания p , содержащие вершины вне поддерева $T_i(w)$, равными кластерам разрезания q из третьей группы.

б) Добавим оставшиеся вершины дерева T' , т. е. вершины вне кластера из \hat{w} или вершины одного из кластеров третьей группы разрезания q , к кластеру, содержащему вершину w . Заметим, что эти вершины суть в точности вершины из единственного кластера, представляющего вторую группу разрезания, не находящиеся в поддереве $T_i(w)$.

Ясно, что q не может быть оптимальным, если \hat{w} по величине не равно $w^{(j)}$.

Наконец, оптимальное разрезание T' , кластеры которого образуют связные подграфы, никогда не может быть образовано из некоторого разрезания \bar{w} поддерева $T_i(w)$, если \bar{w} обладает следующими свойствами:

- 1) его кластер, содержащий вершину w , имеет вес j ;
- 2) его величина меньше, чем величина разрезания $w^{(j)}$.

Мы можем тогда игнорировать все такие разрезания при генерировании оптимального разрезания дерева T' .

Рассмотрим число шагов, требуемых для формирования всех возможных разрезов при комбинировании поддеревьев $T_i(w)$ и $T(v)$ на рис. 5.2 для создания разрезов поддерева $T_{i+1}(w)$. Существует самое большее M разрезов дерева $T_{i+1}(w)$, представляющих множество оптимальных разрезов $\tilde{w}^{(j)}$. Каждое из таких разрезов выбирается из максимум j разрезов $C[w_a, v_b]$, где $a + b = j$ и $a \geq 1$, $b \geq 0$. Наибольшее число разрезов, которые должны быть рассмотрены при формировании $\tilde{w}^{(j)}$, определяется суммой $1 + 2 + 3 + \dots + M = M(M + 1)/2$.

При формировании оптимального разрезания дерева T' требуется скомбинировать разрезания некоторого поддерева $T_i(w)$ с разрезаниями некоторого поддерева $T(v)$ по одному разу для каждой дуги в дереве. Так как дерево с n вершинами имеет $n - 1$ ребер, вычислительная сложность алгоритма пропорциональна M^2n и не зависит от структуры дерева.

Пример. Пусть дерево, показанное на рис. 5.4, а, требуется разрезать на кластеры, вес которых не превышает трех. Вес каждой вершины равен одному, веса

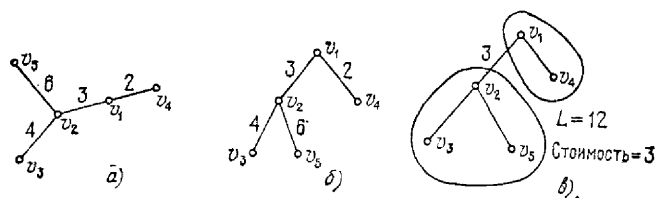


Рис. 5.4.

ребер указаны на рисунке. Ориентированное упорядоченное дерево T' , получающееся из дерева T , показано на рис. 5.4, б. Проиллюстрируем шаги формирования оптимального разрезания в соответствии с алгоритмом.

Начальные значения:

$$1^{(1)} = (v_1), \quad L(1^{(1)}) = 0,$$

$$2^{(1)} = (v_2), \quad L(2^{(1)}) = 0,$$

$$3^{(1)} = 3^{(0)} = (v_3), \quad L(3^{(0)}) = 0,$$

$$4^{(1)} = 4^{(0)} = (v_4), \quad L(4^{(0)}) = 0,$$

$$5^{(1)} = 5^{(0)} = (v_5), \quad L(5^{(0)}) = 0.$$

Формирование разрезания поддерева с корнем в вершине v_2 :

Первая итерация:

$$\tilde{2}^{(1)} = C[2^{(1)}, 3^{(0)}] = (v_2)(v_3)^* \text{ с величиной } L = 0.$$

Данное разрезание единственное; оно и берется в качестве оптимального разрезания из семейства $C[w_a, v_b] = w^{(j)}$. (В дальнейшем выбираемые разрезания будут отмечаться звездочкой.)

$$\tilde{2}^{(2)} = \begin{cases} C[2^{(1)}, 3^{(1)}] = (v_2, v_3)^* \text{ с величиной } = 4, \\ C[2^{(2)}, 3^{(0)}] \text{ не существует, так как} \\ \text{не определено } 2^{(2)}. \end{cases}$$

После завершения первой итерации полагаем:

$$2^{(1)} = (v_2)(v_3), \quad 2^{(2)} = (v_2, v_3).$$

Вторая итерация:

$$\begin{aligned}\tilde{2}^{(1)} &= C[2^{(1)}, 5^{(0)}] = (v_2)(v_3)(v_5)^* \text{ с величиной } = 0; \\ \tilde{2}^{(2)} &= \begin{cases} C[2^{(1)}, 5^{(1)}] = (v_2, v_5)(v_3)^* \text{ с величиной } = 6, \\ C[2^{(2)}, 5^{(0)}] = (v_2, v_3)(v_5) \text{ с величиной } = 4; \\ C[2^{(1)}, 5^{(2)}] \text{ не существует, так как} \\ \hspace{15em} 5^{(2)} \text{ не определено,} \end{cases} \\ \tilde{2}^{(3)} &= \begin{cases} C[2^{(2)}, 5^{(1)}] = (v_2, v_3, v_5)^* \text{ с величиной } = 10, \\ C[2^{(3)}, 5^{(0)}] \text{ не существует, так как} \\ \hspace{15em} 2^{(3)} \text{ не определено.} \end{cases}\end{aligned}$$

После завершения второй итерации полагаем:

$$\begin{aligned}2^{(0)} &= (v_2, v_3, v_5), \quad L(2^{(0)}) = 10, \\ 2^{(1)} &= (v_2)(v_3)(v_5), \quad L(2^{(1)}) = 0, \\ 2^{(2)} &= (v_2, v_5)(v_3), \quad L(2^{(2)}) = 6, \\ 2^{(3)} &= 2^{(0)}, \quad L(2^{(3)}) = 10.\end{aligned}$$

Формирование разрезания поддерева с корнем в вершине v_1 :

Первая итерация:

$$\begin{aligned}\tilde{1}^{(1)} &= C[1^{(1)}, 2^{(0)}] = (v_1)(v_2, v_3, v_5)^* \text{ с величиной } = 10; \\ \tilde{1}^{(2)} &= \begin{cases} C[1^{(1)}, 2^{(1)}] = (v_1, v_2)(v_3, v_5)^* \text{ с величиной } = 3, \\ C[1^{(2)}, 2^{(0)}] \text{ не определено, так как} \\ \hspace{15em} \text{не определено } 1^{(2)}; \end{cases} \\ \tilde{1}^{(3)} &= \begin{cases} C[1^{(1)}, 2^{(2)}] = (v_2, v_5, v_1)(v_3)^* \text{ с величиной } = 9, \\ C[1^{(2)}, 2^{(1)}] \text{ не определено, так как} \\ \hspace{15em} \text{не определено } 1^{(2)}, \\ C[1^{(3)}, 2^{(0)}] \text{ не определено, так как} \\ \hspace{15em} \text{не определено } 1^{(3)}. \end{cases}\end{aligned}$$

После первой итерации полагаем:

$$\begin{aligned}1^{(1)} &= (v_1)(v_2, v_3, v_5), \quad 1^{(2)} = (v_1, v_2)(v_3, v_5), \\ 1^{(3)} &= (v_3, v_5, v_1)(v_3).\end{aligned}$$

Вторая итерация:

$$\begin{aligned} \tilde{1}^{(1)} &= C[1^{(1)}, 4^{(0)}] = (v_1)(v_2, v_3, v_5)(v_4)^* \text{ с величиной } = 10; \\ \tilde{1}^{(2)} &= \begin{cases} C[1^{(1)}, 4^{(1)}] = (v_1, v_4)(v_2, v_3, v_5)^* \text{ с величиной } = 12; \\ C[1^{(2)}, 4^{(0)}] = (v_1, v_2)(v_3, v_5)(v_4) \text{ с величиной } = 3; \end{cases} \\ \tilde{1}^{(3)} &= \begin{cases} C[1^{(1)}, 4^{(2)}] \text{ не определено, так как} \\ \hspace{15em} \text{не определено } 4^{(2)}, \\ C[1^{(2)}, 4^{(1)}] = (v_1, v_2, v_4)(v_3, v_5) \text{ с величиной } = 5, \\ C[1^{(3)}, 4^{(0)}] = (v_1, v_2, v_5)(v_3)(v_4)^* \text{ с величиной } = 9. \end{cases} \end{aligned}$$

После завершения второй итерации полагаем:

$$\begin{aligned} 1^{(1)} &= (v_1)(v_2, v_3, v_5)(v_4), \quad L(1^{(1)}) = 10, \\ 1^{(2)} &= (v_1, v_4)(v_2, v_3, v_5), \quad L(1^{(2)}) = 12, \\ 1^{(3)} &= (v_1, v_2, v_5)(v_3)(v_4), \quad L(1^{(3)}) = 9, \\ 1^{(0)} &= 1^{(2)}. \end{aligned}$$

Поэтому оптимальное разрезание данного дерева (рис. 5.4, в) имеет вид

$$1^{(0)} = (v_1, v_4)(v_2, v_3, v_5), \quad L = 12.$$

1.4. Разрезания графов общего вида. Наложим некоторые ограничения: вершины графа должны иметь неотрицательные целочисленные веса c_i , веса ребер должны быть положительны, все рассматриваемые графы должны быть связны. Для несвязного графа все компоненты связности разрезаются независимо. Последнее ограничение состоит в том, что мультиграфы должны преобразовываться в графы с помощью процедуры: все параллельные ребра между вершинами x и y заменяются одним ребром, вес которого равен сумме весов исходных ребер.

Применим метод динамического программирования. Занумеруем вершины графа в произвольном порядке. Исходя из этой нумерации вершин, будем считать, что j -й шаг процесса разрезания порождает допустимые разрезания подграфа, порожденного всеми теми вершинами, номера которых не больше j . Эти разрезания соответствуют состояниям j -го шага. Разрезания на j -м шаге порождаются разрезаниями, полученными на $j-1$ -м шаге, путем добавления вершины с номером j к этим разрезаниям с учетом весовых ограничений.

Метод решения состоит в определении, к каким кластерам, полученным на $j-1$ -м шаге, может быть добавлена вершина с номером j так, чтобы в итоге были получены допустимые разрезания j -го шага.

Определим k -смежность вершины v_i как допустимое разрезание, порожденное на j -м шаге, которое включает в себя кластер, содержащий вершину v_i и имеющий вес k . Пример 2-смежности вершины v_4 показан на рис. 5.5, а. Рис. 5.5, б показывает, что k -смежность вершины определяется неоднозначно.

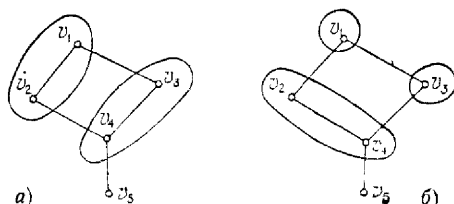


Рис. 5.5.

Допустимым разрезанием графа G называется разрезание, кластеры которого удовлетворяют условиям:

а) сумма весов вершин в кластере не превосходит весового ограничения M ;

б) вершины кластера порождают связный подграф.

В процессе разрезания связного графа будут рассматриваться только допустимые разрезания.

Связным множеством для вершины v_i называется такое множество вершин, что если одна или более вершин из этого множества встречается в кластере разрезания, порожденного на $j-1$ -м шаге, то гарантируется, что добавление вершины v_i к такому кластеру может на шаге $m > j$ привести к образованию связного подграфа.

Связное множество для вершины v_i обозначим через $\mathcal{V}(j)$. Вершина v_i , входящая в связное множество $\mathcal{V}(j)$, обладает свойствами:

1. $i < j$.

2. Вершина v_i либо смежна с вершиной v_j , либо лежит на пути $[v_i, v_{t_1}, v_{t_2}, \dots, v_{t_r}, v_j]$, причем $\sum c_{t_l} \leq M$ и $t \equiv \{i, t_1, t_2, \dots, t_r, j\}$, $t_1 > j, t_2 > j, \dots, t_r > j$.

Второе свойство гарантирует, что разрезание с кластером, содержащим две вершины v_i и v_j , которые в настоящий момент не связны, но становятся связны-

ми, если вершины $v_{t_1}, v_{t_2}, \dots, v_{t_r}$ будут добавлены к кластеру, генерируется на j -м шаге.

На рис. 5.6 показаны связные множества вершин графа G .

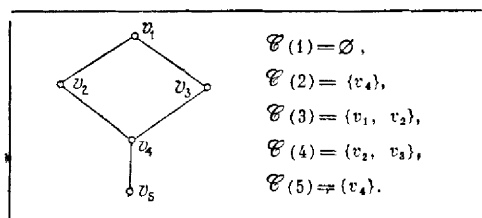


Рис. 5.6.

Опишем алгоритм динамического программирования для формирования оптимального разрезания графа. Эффективность алгоритма зависит от выбора нумерации вершин. $j-1$ -й шаг алгоритма имеет в качестве своих состояний допустимые разрезания подграфа, порожденного вершинами с номерами $1, 2, \dots, j$. Будем обозначать такое множество разрезаний через P_{j-1} . Добавим вершину v_j ко всем разрезаниям из P_{j-1} , которые содержат кластер, удовлетворяющий критерию: добавление вершины v_j не выводит вес кластера за пределы ограничения на вес, и существует вершина в множестве $\mathcal{C}(j)$, входящая в этот кластер.

Как и выше, под величиной разрезания будем понимать сумму весов ребер, входящих в подграф, порожденный вершинами кластеров, причем оба конца каждого из таких ребер должны принадлежать одному кластеру.

Алгоритм:

Вход: Граф $G = (V, E)$, веса c_j вершин v_j , веса ребер.

Выход: Оптимальное разрезание P с кластерами, вес которых не превышает M .

начало

1. для каждой вершины v_j цикл
2. **начало**
3. найти связное множество $\mathcal{C}(j)$;
4. **конец** цикла;
5. $j := 0$; $P_0 := \emptyset$;
6. для j от 0 пока $j < n$ цикл

7. начало

8. образовать k -смежности для $k = c_j$;

примечание каждая такая k -смежность образуется добавлением кластера, содержащего вершину v_j (одну), к множеству кластеров разрезания из P_{j-1}

конец примечания

9. для k от c_j до M цикл

10. начало

11. образовать k -смежности вершины v_j , используя только те разрезания из множества P_{j-1} , которые имеют по крайней мере один кластер с вершиной из множества $\mathcal{C}(j)$;

12. конец цикла;

13. сформировать множество P_j разрезов из построенных k -смежностей;

14. вычислить величины построенных разрезов;

15. конец цикла;

16. выбрать допустимое разрезание в множестве P_j с наибольшей величиной;

конец

Пример использования алгоритма дан на рис. 5.7. Результаты вычислений сведены в таблицу. Каждая строка таблицы соответствует шагу алгоритма; k -й столбец и j -я строка таблицы содержат k -смежности вершины v_j .

Оценим скорость роста требуемого времени при применении алгоритма. Установим для этого скорость роста числа допустимых разрезов для связного графа, так как трудоемкость алгоритма растет пропорционально величине $\sum_{j=1}^n n p_j (\log p_j)$, где $p_j = |P_j|$, а объем требуемой памяти растет пропорционально $n p_j$, где n — число вершин в графе.

Рассмотрим вначале рост мощности множеств P_j . Предположим, что граф полный; предположим также, что ни одно весовое ограничение не нарушается. Верхняя граница для мощности P_j есть число способов, которыми j различных объектов могут быть распределены по i неразличимым ячейкам, где $i \in [1:j]$. Известно, что это число есть число Стирлинга второго рода $S(i, j)$. Таким образом, $p_j = |P_j| < \sum_{i=1}^j S(i, j)$.

Метод вычисления такой суммы неизвестен, но верхняя граница может быть выведена из рекуррентного

Шар	k-смежности		
	1	2	3
1	$(v_1) \quad L=0$		
2	$(v_1)(v_2), L=0$	$(v_1, v_2), L=5$	
3	$(v_1)(v_2)(v_3), L=0$ $(v_1, v_2)(v_3), L=0$	$(v_1, v_3)(v_2), L=3$ $(v_1)(v_2, v_3), L=0$	$(v_1, v_2, v_3), L=8$
4	$(v_1)(v_2)(v_3)(v_4), L=0$ $(v_1, v_2)(v_3)(v_4), L=5$ $(v_1, v_3)(v_2)(v_4), L=3$ $(v_1, v_2, v_3)(v_4), L=8$	$(v_1)(v_2, v_4)(v_3), L=1$ $(v_1)(v_2)(v_3)(v_4), L=6$ $(v_1, v_3)(v_2, v_4), L=4$ $(v_1, v_2)(v_3, v_4), L=11$	$(v_1, v_2, v_4)(v_3), L=6$ $(v_1, v_3, v_4)(v_2), L=9$ $(v_1)(v_2, v_3, v_4), L=7$
5	$(v_1)(v_2)(v_3)(v_4)(v_5), L=0$ $(v_1, v_2)(v_3)(v_4)(v_5), L=5$ $(v_1, v_3)(v_2)(v_4)(v_5), L=3$ $(v_1, v_2, v_3)(v_4)(v_5), L=8$ $(v_1)(v_2, v_4)(v_3)(v_5), L=1$ $(v_1)(v_2)(v_3, v_4)(v_5), L=6$ $(v_1, v_3)(v_2, v_4)(v_5), L=4$ $(v_1, v_2)(v_3, v_4)(v_5), L=11$ $(v_1, v_2, v_4)(v_3)(v_5), L=6$ $(v_1, v_3, v_4)(v_2)(v_5), L=9$ $(v_1)(v_2, v_3, v_4)(v_5), L=7$	$(v_1)(v_2)(v_3)(v_4, v_5), L=4$ $(v_1, v_2)(v_3)(v_4, v_5), L=9$ $(v_1, v_3)(v_2)(v_4, v_5), L=7$ $(v_1, v_2, v_3)(v_4, v_5), L=12$	$(v_1)(v_2, v_4, v_5)(v_3), L=5$ $(v_1)(v_2)(v_3, v_4, v_5), L=10$ $(v_1, v_3)(v_2, v_4, v_5), L=8$ $(v_1, v_2)(v_3, v_4, v_5), L=15$ Оптимальное разрезание $(v_1, v_2)(v_3, v_4, v_5)$ с величиной 15

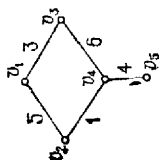


Рис. 7

соотношения $p_j < (1 + t_j)p_{j-1}$, где $t_j = C(j)$. Это вытекает из того факта, что P_j состоит из двух подмножеств: а) 1-смежностей в множестве P_j (их число — p_{j-1}); б) k -смежностей в P_j для $k > 1$. Мощность множества k -смежностей ограничена числом $t_j p_{j-1}$, так как каждая вершина в множестве $\mathcal{C}(j)$ может породить не более чем p_{j-1} разрезов в P_j ; отсюда и из того, что для полного графа $|\mathcal{C}(j)| = j - 1$, следует неравенство $p_j < j!$.

Нижняя граница числа допустимых разрезов рассчит, как f^j , где $1 < f < 2$; это вытекает из решения задачи для дерева.

Итак, трудоемкость и требуемая память алгоритмов, основанных на динамическом программировании, растут экспоненциально с ростом числа вершин графа, ограничивая полезность этих алгоритмов.

1.5. Использование свойств разрезаемых графов. Первое свойство касается изолированных множеств. Используя это понятие, покажем, что рост числа разрезов, порождаемых на шаге j , зависит только от числа вершин с номерами, меньшими j , связанных с вершинами, чьи номера больше или равны j , и не зависит от j . Второе свойство связано с наличием в графе точек сочленения.

Изолированным множеством $I(j)$ для вершины v_i называется множество вершин $\{v_i\}$, удовлетворяющих условиям:

а) $i < j$;

б) вершина v_i не смежна ни с одной вершиной с номером, большим j .

На рис. 5.8 приведены изолированные множества для вершин графа G .

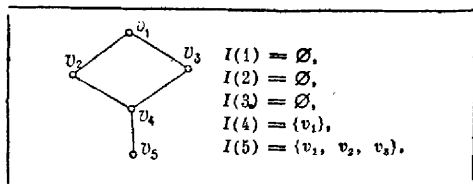


Рис. 5.8.

Лемма 1. *Мощность множества $I(j)$ не зависит от весового ограничения.*

Лемма 2. *Связное множество и изолированное множество взаимно исключают друг друга.*

Лемма 3. Пусть $\mathcal{V}(j)_{\max}$ обозначает множество вершин с номерами, меньшими j , не являющимися элементами множества $I(j)$. Тогда $\mathcal{V}(j)_{\max} = \{v_i | i = 1, \dots, j-1\} - I(j)$ и не зависит от весового ограничения.

Лемма 4. $\mathcal{V}(j)_{\max} \equiv \mathcal{V}(j)$ для каждого весового ограничения M .

Заметим, что $\mathcal{V}(j)$ зависит от M , а $\mathcal{V}(j)_{\max}$ не зависит.

Докажем, что мощность множества $I(j)$ есть неубывающая функция от j .

Теорема 1. $I(j) \subseteq I(j+1)$.

Доказательство. Предположим, что $I(j) \not\subseteq I(j+1)$. Тогда найдется по крайней мере одна вершина v_i , которая принадлежит множеству $I(j)$, но не принадлежит $I(j+1)$. По определению изолированного множества вершина v_i не смежна с вершиной, номер которой больше j ; следовательно, она не смежна с вершиной, номер которой больше $j+1$, что противоречит предположению. Поэтому $I(j) \subseteq I(j+1)$.

Покажем, как понятие изолированного множества может быть использовано для модификации процесса разрезания.

Выше мы определили $\mathcal{V}(j)_{\max}$ как множество тех вершин с номерами $1, 2, \dots, j-1$, которые не входят в множество $I(j)$. Разрезания, порождаемые на $j-1$ -м шаге, могут быть разделены на непересекающиеся подмножества, в каждое из которых входят только те разрезания, которые имеют одно и то же распределение вершин из $\mathcal{V}(j)_{\max}$ по кластерам разрезания. Более того, каждый кластер, содержащий одну или более вершин из $\mathcal{V}(j)_{\max}$, имеет тот же самый вес, что и соответствующий ему кластер из другого разрезания, но из того же множества разрезаний. Примером таких разрезаний могут служить разрезания $(v_1, v_2)(v_4)(v_3, v_5)$ и $(v_1)(v_2)(v_4)(v_3, v_5)$ из одного и того же множества, где $\mathcal{V}(6)_{\max} = \{v_4, v_5\}$ и каждая вершина имеет единичный вес. Назовем два разрезания *подобными*, если они принадлежат одному и тому же множеству разрезаний. *Доминантным* разрезанием множества подобных разрезаний назовем разрезание наибольшей величины. Если имеются два или более подобных разрезаний с одним и тем же наибольшим весом, то в качестве доминантного берется одно из них.

Причина введения доминантных разрезаний состоит в том, что все разрезания из множества разрезаний

P_{j-1} , генерируемых на $j-1$ -м шаге (кроме доминантных), могут быть исключены из дальнейшего рассмотрения. Этот результат снижает верхнюю оценку числа допустимых разрезов, генерируемых на j -м шаге, с $j!$ до $\hat{t}_j(t_j!)M^{\hat{t}_j}$, где $\hat{t}_j = |\mathcal{C}(j)_{\max}|$. Для малых значений M и \hat{t}_j происходит значительное снижение числа допустимых разрезов.

Теорема 2. Для генерирования разрезов на $k-1$ -м шаге необходимы только доминантные разрезы, полученные на k -м шаге.

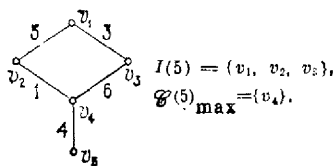
Доказательство. Пусть G есть n -вершинный граф. Разрезание p графа G на некотором шаге k может быть представлено в виде последовательности пар $[v_1, ()]$, $[v_2, X_2]$, $[v_3, X_3]$, ..., $[v_k, X_k]$, в которых первый элемент есть вершина с номером $i \leq k$, а второй — кластер, к которому вершина v_i была добавлена на i -м шаге ($()$ обозначает пустой кластер). Преимущество такого представления состоит в том, что оно полностью описывает, как было порождено разрезание p . Например, разрезание $[v_1, ()]$, $[v_2, (v_1)]$, $[v_3, (v_2)]$, $[v_4, (v_1)]$, $[v_5, (v_2, v_3)]$ в обычной записи имеет вид $p = (v_1, v_4) \times \times (v_2, v_3, v_5)$.

Пусть P_i — множество разрезов, порождаемых на i -м шаге. Определим теперь образующую разрезов p из разрезания q , где p берется из P_k , а q — из P_j , $j < k$, как последовательность $[v_{j+1}, X_{j+1}]$, $[v_{j+2}, X_{j+2}]$, ..., $[v_k, X_k]$. Это обозначение есть вариант использованного выше представления разрезания p , при котором игнорируются шаги, ведущие к образованию разрезания q .

Пусть разрезания f и g , порожденные на $k-1$ -м шаге, подобны, и пусть f есть доминантное разрезание, а g таковым не является. Предположим, что существует разрезание графа G , например g_n , полученное из g , величина которого больше величины любого разрезания графа G , полученного из f , и покажем, что это неверно. Пусть образующая g_n из g есть $[v_k, X_k]$, $[v_{k+1}, X_{k+1}]$, ..., $[v_n, X_n]$. Так как f и g подобны, то существует разрезание f_n , получаемое из f , с образующей $[v_k, \bar{X}_k]$, $[v_{k+1}, \bar{X}_{k+1}]$, ..., $[v_n, \bar{X}_n]$ такой, что для $i \in [k : n]$ кластеры X_i и \bar{X}_i имеют один и тот же вес и вершины в кластере X_i отличаются от вершин в кластере \bar{X}_i только в том случае, если они принадлежат множеству $I(k)$. Так как вершины в изолированном

множестве для вершины v_k не соединены ребром ни с одной вершиной, чей номер меньше чем $k-1$, то величина каждого разрезания, порожденного на шагах $k, k+1, \dots, n$, не зависит от вершин из множества $I(k)$, которые встречаются в кластерах вместе с вершинами из множества $\mathcal{C}(k)_{\max}$.

Так как кластеры X_i и $\bar{X}_i, i \in [k:n]$, имеют вершины, различающиеся только тогда, когда они принадлежат множеству $I(k)$, сумма весов ребер в X_i и \bar{X}_i может отличаться на сумму весов тех ребер, которые соединяют вершины из множества $I(k)$, входящие в кластеры. Так как f есть доминантное разрезание (f доминирует g), то сумма весов ребер в кластере \bar{X}_i равна или больше суммы весов ребер в кластере X_i и разрезание f_n доминирует разрезание g_n . Следовательно, величина f_n больше или равна величине g_n . В силу



Множества подобных разрезаний в P_4 (см. рис. 5.7):

S_1	S_2
$(v_1)(v_2)(v_3)(v_4) \quad L = 0,$	$(v_1)(v_3)(v_2, v_4), \quad L = 1,$
$(v_1, v_2)(v_3)(v_4), \quad L = 5,$	$(v_1)(v_2)(v_3, v_4), \quad L = 6,$
$(v_1, v_3)(v_2)(v_4), \quad L = 3,$	$(v_1, v_3)(v_2, v_4), \quad L = 4,$
$(v_1, v_2, v_3)(v_4), \quad L = 8,$	$(v_1, v_3)(v_3, v_4), \quad L = 11,$

S_3

$(v_3)(v_1, v_2, v_4), \quad L = 6,$
$(v_2)(v_1, v_3, v_4), \quad L = 9,$
$(v_1)(v_2, v_3, v_4), \quad L = 7,$

Доминантные разрезания в P_4 :

в S_1 : $(v_1, v_2, v_3)(v_4), \quad L = 8,$
в S_2 : $(v_1, v_2)(v_3, v_4), \quad L = 11,$
в S_3 : $(v_2)(v_1, v_3, v_4), \quad L = 9.$

Рис. 5.9.

этого удаление всех разрезаний из множества разрезаний P_{k-1} , доминируемых каким-либо разрезанием, повышает эффективность процесса. Теорема доказана.

Пример, иллюстрирующий теорему, приведен на рис. 5.9.

Заметим, что мощность изолированного множества для вершин графа есть функция нумерации вершин. Однако техника нумерации, минимизирующая число допустимых разрезов, не известна.

Выше мы ввели понятия разделяющей вершины и блока в графе и дали алгоритм их отыскания. Если в связном графе имеется более одного блока, то имеет смысл находить оптимальные разрезы каждого блока в отдельности в любом порядке и потом комбинировать их для получения оптимального разреза всего графа.

Теорема 3. Если граф G имеет q блоков, $q > 1$, то оптимальное разрезание p_{opt} графа G может быть получено отысканием вначале оптимальных разрезов его блоков и последующим комбинированием полученных разрезов.

Доказательство. Рассмотрим вершинное представление разрезания p_{opt} :

$$\underbrace{[() \dots ()]}_{NC_1} \underbrace{[() \dots ()]}_{NC_2} \dots \underbrace{[() \dots ()]}_{NC_q} \underbrace{[() \dots ()]}_C,$$

где NC_i есть (возможно, пустое) множество кластеров, вершины которых не являются точками сочленения и все они принадлежат одному и тому же блоку B_i . Множество C состоит из кластеров, каждый из которых содержит по крайней мере одну точку сочленения. Вершинное представление p_{opt} допускает такую форму из-за специального свойства графа с блоками. Так как единственной вершиной в блоке B_i , смежной с вершинами вне блока, является точка сочленения, кластер, который содержит вершины из B_i , но не содержит точки сочленения, должен содержать исключительно вершины из B_i в силу ограничения на связность подграфа, порожденного вершинами кластера. Это свойство оправдывает подразбиение NC_i .

Каждый кластер $X \in C$ содержит два типа вершин:

1. Множество точек сочленения $\{v_{k_1}, \dots, v_{k_t}\}$.

2. Множество вершин $\{v_{i_1}, v_{i_2}, \dots, v_{i_a}, v_{j_1}, v_{j_2}, \dots\}$,

ни одна из которых точкой сочленения не является.

Последнее множество может быть разбито на подмножества, соответствующие классам эквивалентности отношения БЛОК, где u БЛОК v тогда и только тогда, когда u и v — вершины из одного блока. Если исключить ограничение на расщепление вершин, то кластер

X может быть заменен множеством кластеров $\{X_1, \dots, X_T\}$, каждый из которых обладает следующими свойствами:

1. Каждый кластер содержит объединение множества вершин из некоторого блока B_j , порождаемого отношением эквивалентности БЛОК, и множества точек сочленения блока B_j .

$$2. \sum_{i=1}^T L[X_i] = L[X], \quad L[X_i] = \sum_{v_\alpha, v_\beta \in X_i} c_{\alpha\beta}.$$

Заметим, что одна и та же точка сочленения может встретиться в нескольких кластерах из множества $\{X_1, \dots, X_T\}$.

Когда мы проведем описанный выше процесс над каждым кластером в C , вершинное представление разрезания p_{opt} преобразуется в

$$\underbrace{[(\) \dots (\)]}_{NC_1} \dots \underbrace{[(\) \dots (\)]}_{NC_q} \underbrace{[(\) \dots (\)]}_{C_1} \dots \underbrace{[(\) \dots (\)]}_{C_q},$$

где C_i — множество кластеров, вершины которых принадлежат блоку B_i и которые содержат по крайней мере одну точку сочленения блока B_i . Величина разрезания от этого не изменяется, а само разрезание p_{opt} представляется теперь множествами кластеров (NC_i, C_i) , соответствующих разрезаниям блока B_i . Благодаря расщеплению точек сочленения не существует ребер из кластера из множества (NC_i, C_i) в кластер из множества (NC_j, C_j) для $i \neq j$.

Заметим, что можно обратить процесс перехода от p_{opt} к p'_{opt} и генерировать p_{opt} , порождая вначале разрезания блоков, а затем комбинируя их.

Наличие точек сочленения и, следовательно, блоков в графе уменьшает число разрезов, порождаемых на каждом шаге, до числа, прямо пропорционального числу разрезов, генерируемых в каждом блоке, если блок разрезается независимо от других. Кратко алгоритм разрезания графа с блоками может быть описан следующим образом.

Предположим, что граф G имеет q блоков B_1, \dots, B_q . Пусть блок B_k имеет b_k точек сочленения. Разрезаем каждый блок независимо, выбрасывая точки сочленения из каждого изолированного множества для вершин

в B_k и сохраняя оптимальные 1-смежности, 2-смежности, ..., M -смежности для каждой точки сочленения в блоке B_k . Поэтому для блока B_k нужно хранить не более $b_k M$ множеств.

Когда все блоки разрезаны, мы находим блок с одной точкой сочленения и комбинируем его с блоком, содержащим ту же точку сочленения. Делаем это, склеивая кластеры, содержащие точку сочленения, и оставляя остальные кластеры неизменными. Эти операции требуют не более чем $M(M+1)/2$ шагов.

Склеивание двух блоков может уменьшить число несклеенных точек сочленения в результирующем подграфе, содержащем эти два блока, на единицу. Найдём далее другой блок с одной несклеенной точкой сочленения и склеиваем его с блоком, содержащим их общую точку сочленения. Продолжаем этот процесс, пока не будет найдено оптимальное разрезание всего графа. Так как граф блоков $F(G)$ графа G есть дерево, следовательно, метод склеивания блоков при разрезании графов есть модификация описанного выше метода разрезания деревьев.

Используя проведенный анализ, можно показать, что оценка $p_j = |P_j| < \hat{t}_j (\hat{t}_j!) M^{\hat{t}_j} < j!$ для n -вершинного дерева превращается в $p_j = |P_j| < M$, где M — константа весового ограничения, так как для дерева $\hat{t}_j = |\mathcal{E}(j)_{\max}| = 1$ для всех $j > 1$.

Еще один граф, у которого величина \hat{t}_j не зависит от j , показан на рис. 5.10. Для него $I(j) = \{v_i | i < j - h\}$, где h — параметр, определяющий «ширину» графа. Таким образом, имеем $\hat{t}_j = h$ для всех j и $p_j = |P_j| < h(h!)M^h$. Более тонкий анализ даст оценку $p_j = |P_j| < M^h$.

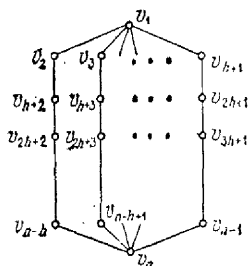


Рис. 5.10.

4.6. Для разрезания встречающихся на практике графов с сотнями и тысячами вершин целесообразно использовать приближенные методы, основанные на идеях поэлементного построения

кластеров — последовательные алгоритмы — и на идеях итерационной перекомпоновки кластеров по некоторым критериям с целью минимизации суммарных межкластерных связей — итерационные алгоритмы.

Один из эвристических алгоритмов разрезания произвольного графа можно построить на базе алгоритма разрезания деревьев. В основе его лежит построение каркаса наибольшего возможного веса, разрезание его и дополнение разрезов каркаса до разрезов всего графа. На рис. 5.11 приведен пример такого разрезания.

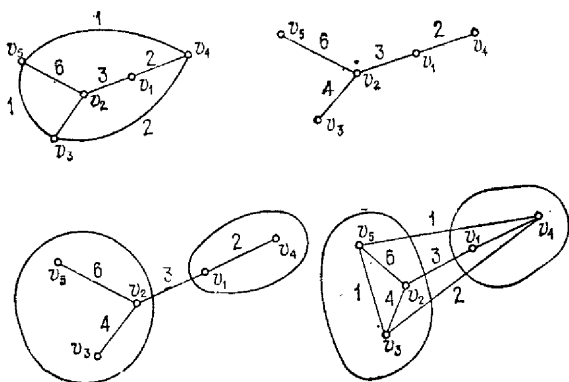


Рис. 5.11.

Рассмотрим более тонкий алгоритм субоптимального разрезания графа, основанный на структурированном методе ветвей и границ. Алгоритм структурируется в виде семерки $A = (R, B, L, F, D, S, E)$, где R — правило представления разрезания, B — правило ветвления, L — функция характеристики разрезания, F — граничная функция, D — правило предпочтения разрезания, S — правило выбора разрезания, E — правило исключения разрезов. Как и в методе динамического программирования, мы используем имеющуюся аналогию с задачей о рюкзаке и приближенный метод ее решения.

Возьмем кластер X_k разрезания $R(X) = (X_1, \dots, X_r)$ графа $G = (X, U)$ в качестве рюкзака с исходной емкостью m_k , равной количеству вершин в кластере, а вершины и ребра других кластеров — в качестве предметов, подлежащих загрузке в рюкзак. По методу приближенного решения задачи о рюкзаке определяем вершины, загружающие кластер X_k с некоторым избытком. В результате такой операции загрузки остальные кластеры будут испытывать недостаток в вершинах. Затем, считая вершины и ребра подграфа $\langle X_k \rangle$ пред-

метами, а кластеры с недостающими вершинами — рюкзаками, решаем задачу о рюкзаке для каждого кластера в порядке убывания дефицитов вершин (разгрузка).

В итоге операции загрузки и разгрузки (*RR*-операции) получаем новое разрезание. *RR*-операция повторяется для кластеров X_k , $k \in [1:r]$, в порядке, определяемом некоторой дополнительной информацией о разрезании. Исходное разрезание также выбирается при помощи *RR*-операции. Такой подход позволяет выделить сложное комбинаторное ядро задачи разрезания и применить к нему субоптимальный алгоритм ветвления, довольно простой по структуре и эффективный по количеству перебираемых вариантов. Трудоемкость алгоритма $\sim O(n^3) \div O(n^4)$, что позволяет использовать его для разрезания графов, содержащих до тысячи вершин.

Уточним содержание компонент семерки, описывающей алгоритм по схеме структурированного метода ветвей и границ:

$$A = (R, B, L, F, D, S, E). \quad (5.7)$$

Компоненты этой семерки определяются следующим образом:

1°. Правило *R* представления состояния из множества возможных представлений разрезаний для задачи определяет наиболее рациональное.

2°. Правило *B* ветвления определяет процесс ветвления, целью которого является разбиение множества состояний на непустые подмножества.

3°. Функция *L* характеристики состояния ставит в соответствие каждому состоянию действительное число.

4°. Граничная функция *F* представляет собой экстремальное значение для характеристики состояния.

5°. Правило *D* предпочтения состояний определяет приоритет активных (т. е. подлежащих дальнейшему рассмотрению) состояний при ветвлении.

6°. Правило *S* выбора состояния определяет из множества активных состояний то, которое подлежит ветвлению или является субоптимальным.

7°. Правило *E* исключения состояний отбрасывает неперспективные состояния в смысле их нахождения на пути к субоптимальному решению или с точки зрения объема памяти ЭВМ.

Компоненты семерки (5.7) являются этапами неявного алгоритма по данной схеме, которые при постро-

ении алгоритма циклически повторяются, за исключением правила R , до получения субоптимального решения. Такое решение указывает правило S , используя результаты вычисления функций L и F . Алгоритм решает задачу (5.1)–(5.3). Для удобства изложения введем следующие обозначения:

$$I(X_k) = \sum_{x_i, x_j \in X_k} c_{ij} \text{ — связность кластера } X_k;$$

$$I(R) = \sum_k I(X_k) \text{ — связность разрезания } R(X).$$

Особенностью данного алгоритма является построение в рассматриваемой схеме эффективной функции ветвления B , которая использует RR -операцию. Это позволяет надеяться на получение хорошего приближения к оптимальному решению. Алгоритм строится последовательным определением компонент в (5.7).

Шаг 1. Разрезание $R_i(X)$ графа G будем представлять в виде $R_i(X) = (X_1^i, \dots, X_r^i)$. Исходное разрезание $R_0(X)$ выбирается по указанной на шаге 2 процедуре.

Шаг 2. Правило ветвления B для алгоритма указывает способ получения из $R_0(X)$ новых разрезаний $R_i(X)$, используя для ветвления RR -операцию.

Шаг 2.1. Загрузка RR -операции. Пусть имеется некоторое исходное разрезание $R_0(X) = (X_1^0, \dots, X_r^0)$, причем $I(X_k^0) \geq I(X_{k+1}^0)$ для всех k , $k \in [1, r-1]$. Рассматриваем кластер X_k^0 ($k \in [1, r]$) как рюкзак с исходной емкостью m_k . Вершины и ребра (элементы) подграфа $G_k = (X \setminus X_k^0, U \setminus U_k^0)$, где $U_k^0 = \{(v_i, v_j) \mid v_i, v_j \in X_k^0\}$, считаем предметами, подлежащими загрузке в рюкзак. Среди них по методу приближенного решения задачи о рюкзаке выбирается α вершин, которые максимально, согласно предлагаемому критерию, увеличат связность $I(X_k^0)$ кластера X_k^0 . Ясно, что $\alpha \in \left[1, \left(\sum_{s=1}^r m_s\right) - 1\right]$, $s \neq k$. Будем изображать элементы графа G вектором $Y = (y_l) = (y_i, y_{ij})$, $1 \leq l \leq n(n+1)/2$, где y_i соответствует вершине v_i , $i \in [1:n]$, y_{ij} соответствует ребру (v_i, v_j) , $1 \leq i < j \leq n$, и

$$y_l = \begin{cases} 1, & \text{если элемент } l \text{ подлежит загрузке;} \\ 0 & \text{в противном случае.} \end{cases}$$

Вектору элементов графа, согласно задаче о рюкзаке,

поставим в соответствие вектор весов $A = (a_l) = (a_i, a_{ij})$, где a_i — вес вершины v_i при загрузке, a_{ij} — вес ребра (v_i, v_j) при загрузке, и вектор размерностей $P = (p_l) = (p_i, p_{ij})$, где

$$p_l = \begin{cases} 1, & \text{если } 1 \leq l \leq n; \\ 2, & \text{если } n < l \leq n(n+1)/2. \end{cases}$$

Покажем, как вычислять вектор A . Пусть имеется подграф $(K, V) \subset (X, U)$, и пусть вершина v_i принадлежит множеству $X \setminus K$, а ребро (v_i, v_j) — множеству $U \setminus V$. Обозначим $a_i^+ = I(K \cup \{v_i\}) - I(K)$, $a_i^- = I(K) - I(K - \{v_i\})$, $a_{ij}^+ = I(K \cup \{v_i, v_j\}) - I(K)$, $a_{ij}^- = I(K) - I(K - \{v_i, v_j\})$. Тогда $a_i = a_i^+ - a_i^-$, $i \in [1:n]$, $a_{ij} = a_{ij}^+ - a_{ij}^-$, $1 \leq i < j \leq n$, где $a_{ij}^+ = a_i^+ + a_j^+ + a_{ij}$, $a_{ij}^- = a_i^- + a_j^- - a_{ij}$, $K = X_k^0$, $V = U_k^0$.

Теперь задача о рюкзаке для загрузки кластера X_k^0 до $m_k + \alpha$ вершин будет иметь вид

$$\left. \begin{aligned} AY' &\rightarrow \max, \\ PY' &= m_k + \alpha, \end{aligned} \right\} \quad (5.8)$$

где векторы P и A вычисляются по приведенным выше формулам при $K = X_k^0$ и $V = U_k^0$. Загружаемый элемент определяется по первому неравенству в формуле

$$a_1/p_1 \geq a_2/p_2 \geq \dots \geq a_l/p_l, \quad 1 \leq l \leq n(n+1)/2. \quad (5.9)$$

В зависимости от вида загружаемого элемента емкость кластера увеличивается на единицу (в случае вершины) и на два (в случае ребра). Если кластер не заполнен, то недостающие элементы определяются применением (5.9) для оставшихся элементов. Если кластеру требуется лишь вершина, то соотношения (5.9) достаточно получить для вершин.

Результатом решения задачи (5.8) (или результатом загрузки RR -операции) будет некоторое промежуточное разрезание

$$R_{k(\text{пр})}(X) = (X_1^h(\Delta_1), \dots, X_{k(\text{пр})}^h, \dots, X_r^h(\Delta_r)),$$

где $X_s^h(\Delta_s)$ — кластер X_s^h без Δ_s вершин, загруженных в кластер $X_{k(\text{пр})}^h$ ($k \neq s$), $X_{k(\text{пр})}^h$ — кластер X_k^h без увеличенной емкости $m_k + \alpha$, $\sum_{s=1}^r \Delta_s = \alpha$ ($s \neq k$).

Шаг 2.2. Разгрузка RR -операции. Пусть

$$\Delta_1 \geq \Delta_2 \geq \dots \geq \Delta_r. \quad (5.10)$$

Считаем элементы подграфов с вершинами $N_d = X_{k(\text{пр})}^h \setminus \sum_{s=1}^d X^h(\Delta_s)$ (где $d \in [1:r]$, $d \neq k$, $X^h(\Delta_k) = \emptyset$ и $X^h(\Delta_s)$ — множество вершин, загруженных в кластер $X_s^h(\Delta_s)$) предметами для загрузки соответственно в кластеры $X_s^h(\Delta_s)$, $s \in [1:r]$, $s \neq k$, требующие по Δ_s вершин. Аналогичным образом для каждого кластера $X_s^h(\Delta_s)$, $s \in [1:r]$, $s \neq k$, в порядке, указываемом соотношением (5.10), решаем задачу о рюкзаке

$$\left. \begin{aligned} AY' &\rightarrow \max, \\ PY' &= m_s \end{aligned} \right\}$$

при $K = N_s$. В результате разгрузки получим новое разрезание $R_k(X) = (X_1^h, X_2^h, \dots, X_r^h)$.

Повторим RR -операцию для $k \in [1:r]$ и для α , выбираемого из промежутка $\left[1, \left(\sum_{s=1}^r m_s\right) - 1\right]$, $s \neq k$, с некоторым шагом h . Исходное разрезание выбирается процедурой, которая представляет собой последовательное решение следующих задач о рюкзаке:

$$\left. \begin{aligned} AY' &\rightarrow \max, \\ PY' &= m_j, \quad j = 1, \dots, r-1, \end{aligned} \right\}$$

при $N_1 = X$, $N_j = X \setminus \sum_{i=2}^j X_i$, $j \in [2:r-1]$, а рюкзаками являются первоначально пустые кластеры X_j , требующие для загрузки m_j вершин. В результате получим разрезание $R_0(X) = (X_1^0, \dots, X_r^0)$.

Шаг 3. Функция характеристики разрезания L определяется следующим образом: $L(R_i(X)) = I(R_i)$.

Шаг 4. Граничная функция F есть функция $F(R_i(X)) = \max_{R_i(X)} I(R_i)$.

Шаг 5. Правило предпочтения D говорит о том, что ветвление исходит от $R_0(X)$.

Шаг 6. Правило выбора S очередного разрезания: разрезание $R_{\text{суб}}$ является субоптимальным, если $I(R_{\text{суб}}) = \max_{j \in H} I(R_j)$, где H — множество индексов

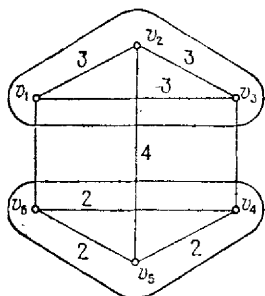


Рис. 5.12.

всех полученных с помощью RR -операции разрезаний.

Шаг 7. Правило исключения разрезаний E для алгоритма говорит о том, что исключаются все разрезания $R_j(X)$, для которых $I(R_j) < I(R_{\text{суб}})$.

Пример. Рассмотрим применение алгоритма на примере разрезания графа, показанного на рис. 5.12. Пусть $r = 2$ и $m_1 = m_2 = 3$. Положим $\alpha = 1$, $h = 0$. Результаты последовательных вы-

числений приведены в табл. 5.1.

Таблица 5.1

Номер разрезания $R_i(X)$	$I(R_i)$	$R_{i(\text{пр})}(X)$	Результат разгрузки RR -операции	Результат разгрузки RR -операции	
				X_1	X_2
0	13			(v_2, v_3, v_4)	(v_1, v_5, v_6)
1	15	$X_{1(\text{пр})}^1$ $X_{2(\Delta_2)}^1$	(v_1, v_2, v_3, v_4) (v_5, v_6)	(v_1, v_2, v_3)	(v_4, v_5, v_6)
2	15	$X_{1(\Delta_1)}^2$ $X_{2(\text{пр})}^2$	(v_3, v_4) (v_1, v_2, v_5, v_6)	(v_1, v_2, v_3)	(v_4, v_5, v_6)

Субоптимальное решение есть разрезание

$$R_{\text{суб}} = R_1(X) = R_2(X) = ((v_1, v_2, v_3), (v_4, v_5, v_6)),$$

которое, кстати, является и оптимальным.

1.7. Разрезание бесконтурного орграфа на минимальное число кластеров. Метод основан на сведении задачи о разрезании к задаче целочисленного программирования. Сложность задачи существенно зависит от ограничения на размер кластера. Занумеруем вершины орграфа $G = (X, \Gamma)$ произвольным способом x_1, x_2, \dots, x_n и построим матрицу смежности $A(G)$.

Теорема 4. Минимальное число $\rho_M(G)$ кластеров, на которое можно разбить вершины орграфа G с соблюдением условий:

$$а) \Gamma X_j \subseteq \bigcup_{i>j} X_i, \quad j = 1, \dots, h, \quad X_{h+1} = \emptyset;$$

$$б) |X_i| \leq M,$$

равно $\rho_M(G) = \sum_{q=1}^n X_q \left(\sum_{i=1}^n \xi_{qi} \right)$, где ξ_{qi} — целые числа, минимизирующие линейную форму

$$L = \sum_{q=1}^n \sum_{i=1}^n r_q \xi_{qi}, \quad (5.11)$$

$$r_1 = 1, \quad r_{q+1} = r_q + (n - q - 1)(r_q - 1) + 1, \quad (5.12)$$

и удовлетворяющие линейным неравенствам и равенствам:

$$а) \quad \xi_{qi} \geq 0;$$

$$б) \quad \sum_{q=1}^n \xi_{qi} = 1; \quad (5.13)$$

$$в) \quad \sum_{i=1}^n \xi_{qi} \leq M;$$

$$г) \quad \sum_{k=1}^n a_{ik} \xi_{qk} \leq M \sum_{s=1}^{q-1} \xi_{si}, \quad i, q = 1, \dots, n.$$

Кластер X_q при этом определяется как множество

$$X_q = \{x_i \in X \mid \xi_{qi} = 1\}. \quad (5.14)$$

Доказательство. Пусть целые числа ξ_{qi} ($q, i \in [1:n]$) удовлетворяют системе (5.13) и минимизируют линейную форму (5.11). В силу неравенства а) и равенства б) из (5.13) каждое число ξ_{qi} будет либо 0, либо 1. Определим множества X_q , $q \in [1:n]$, формулой (5.14) и выберем среди них непустые. Если число их равно h , то это обязательно будут кластеры X_1, \dots, X_h , $h \in [1:n]$, так как иначе ξ_{qi} не минимизировали бы форму (5.11), поскольку из (5.12) следует, что $r_{q+1} > r_q$, $q \in [1:n-1]$.

Покажем, что кластеры X_1, \dots, X_h образуют минимальное разрезание орграфа, удовлетворяющее условиям теоремы. В силу равенства б) для каждого i , $i \in [1:n]$, существует одно и только одно $q \in [1:h]$ такое, что $\xi_{qi} = 1$. Следовательно, каждая вершина x_i попадает в единственный кластер X_q . Из неравенства

в) вытекает, что число вершин в каждом кластере X_q не превышает M .

Пусть $x_j \in \Gamma x_i$, а $x_i \in X_k$. Покажем, что $x_j \in X_q$, где $q > k$. Для этого предположим, что $x_j \in X_q$, $q \leq k$. Имеем $\xi_{ki} = 1$, $\xi_{qj} = 1$. Возьмем в совокупности неравенств г) неравенство $\sum_{h=1}^n a_{ih} \xi_{qh} \leq M \sum_{s=1}^{q-1} \xi_{si}$. При $q \leq k$ правая часть этого неравенства будет равна 0. Левая же часть его будет больше нуля, так как $a_{ij} = 1$ и $\xi_{qj} = 1$. Следовательно, при нашем допущении числа ξ_{qi} не удовлетворяют системе (5.13), и мы заключаем, что $x_j \in X_q$, $q > k$, т. е. что $\Gamma X_k \subseteq \bigcup_{j>k} X_j$.

Остается показать, что разрезание, определяемое кластерами X_1, \dots, X_h , будет минимальным разрезанием, удовлетворяющим условиям теоремы.

Пусть $\bar{X}_1, \dots, \bar{X}_p$ ($p = \rho_M(G)$) — кластеры некоторого минимального разрезания орграфа G , удовлетворяющего условиям теоремы. Построим по данному разрезанию систему из n^2 чисел $\bar{\xi}_{qh}$, $q, h \in [1:n]$, определяемых равенствами:

$$\bar{\xi}_{qh} = \begin{cases} 1, & \text{если } x_h \in \bar{X}_q, \quad 1 \leq q \leq p; \\ 0, & \text{если } x_h \notin \bar{X}_q, \quad 1 \leq q \leq p; \\ 0, & \text{если } p < q \leq n. \end{cases}$$

Легко увидеть, что неравенства а) — в) справедливы для чисел $\bar{\xi}_{qh}$. Покажем справедливость неравенства г). В самом деле, для любых фиксированных i и q , $i, q \in [1:n]$, существуют две возможности:

$$1) \sum_{s=1}^{q-1} \bar{\xi}_{si} = 0; \quad 2) \sum_{s=1}^{q-1} \bar{\xi}_{si} = 1.$$

Первая из них означает, что $x_i \in \bar{X}_t$, $t \geq q$. Отсюда следует, что если $\bar{\xi}_{qh} = 1$, то $a_{ih} = 0$. Следовательно,

$\sum_{h=1}^n a_{ih} \bar{\xi}_{qh} = 0$, т. е. в случае 1) неравенство г) выполняется. Выполняется оно и в случае 2), так как, очевидно, $\sum_{h=1}^n a_{ih} \bar{\xi}_{qh} \leq M$.

Покажем, что предположение $h > p$ ведет к противоречию. Если $h > p$, то числа $\bar{\xi}_{qi}$ доставляют форме (5.11) меньшее значение, нежели система чисел ξ_{qi} .

Действительно, в силу (5.12) имеем

$$\begin{aligned} \sum_{q=1}^n \sum_{i=1}^n r_q \xi_{qi} &\leq r_1 + r_2 + \dots + r_{p-1} + (n-p+1)r_p < \\ &< r_1(n-p) + r_2 + \dots + r_{p+1} \leq \\ &\leq r_1(n-h+1) + r_2 + \dots + r_h \leq \sum_{q=1}^n \sum_{i=1}^n r_q \xi_{qi}. \end{aligned}$$

Это неравенство противоречит тому, что числа ξ_{qi} минимизируют форму (5.11). Следовательно, разрезание, определяемое формулой (5.14), будет минимальным. Теорема доказана.

Покажем теперь, что для класса прогрессивно-конечных графов данная задача решается эффективно.

Граф G называется *прогрессивно-конечным в вершине v_0* , если в графе нет путей бесконечной длины, начинающихся в v_0 . Граф называется *прогрессивно-конечным*, если он прогрессивно-конечен в каждой своей вершине.

Пусть из каждой вершины графа $G = (X, \Gamma)$, $|X| = n$, исходит не более m дуг. Построим множества X_i , $i \in [1:p]$, используя следующую процедуру. Построим последовательность множеств вершин орграфа:

$$\begin{aligned} \tilde{X}_1 &= \{x | x \in X, \Gamma x = \emptyset\}, \\ \tilde{X}_2 &= \{x | x \in X, \Gamma x \subseteq \tilde{X}_1\}, \end{aligned} \tag{5.15}$$

$$\begin{aligned} &\dots \\ \tilde{X}_p &= \{x | x \in X, \Gamma x \subseteq \tilde{X}_{p-1}\}. \end{aligned}$$

Это будет возрастающая последовательность множеств $\tilde{X}_1 \subset \tilde{X}_2 \subset \dots \subset \tilde{X}_j \subset \dots$, стабилизирующаяся при некотором $j = p$. Так как в графе G каждый путь имеет конечную длину, то $\tilde{X}_p = X$.

Построим теперь множества:

$$\begin{aligned} X_1 &= \tilde{X}_p \setminus \tilde{X}_{p-1}, \quad X_2 = \tilde{X}_{p-1} \setminus \tilde{X}_{p-2}, \dots \\ &\dots, X_{p-1} = \tilde{X}_2 \setminus \tilde{X}_1, \quad X_p = \tilde{X}_1. \end{aligned} \tag{5.16}$$

Множества X_j образуют разрезание графа на кластеры, удовлетворяющие условиям $\Gamma X_j \subseteq \bigcup_{i>j} X_i$. Действительно, $X_j = \tilde{X}_{p-j+1} \setminus \tilde{X}_{p-j}$ и $\Gamma \tilde{X}_{p-j+1} \subseteq \tilde{X}_{p-j}$, следовательно, $\Gamma X_j \subseteq \tilde{X}_{p-j} = \bigcup_{i=j+1}^p X_i$, $j \in [1:p]$. Более того, можно

показать, что это разрезание содержит минимально возможное число кластеров.

Предположим теперь, что можно указать такое число $k \in [1: p+1]$, что $X_{p+1} = \emptyset$ и

$$|X_j| \geq m(M-1) + 1, \quad j = 2, 3, \dots, k-1, \quad (5.17)$$

$$|X_i| \leq M, \quad i = k, k+1, \dots, p. \quad (5.18)$$

- Докажем, что при этих предположениях

$$\rho_M(G) = \left\lfloor \frac{1}{M} \sum_{i=1}^{k-1} |X_i| \right\rfloor + p - k + 1. \quad (5.19)$$

Обозначим правую часть равенства (5.19) через H . Покажем вначале, что каков бы ни был граф G , удовлетворяющий условию (5.18) при наименьшем k , для него справедливо неравенство $\rho_M(G) \geq H$. В самом деле, пусть $\sigma_1, \dots, \sigma_t$ — кластеры некоторого разрезания графа G , удовлетворяющего условиям теоремы 4.

Воспользовавшись неравенством $\sum_{i=t-p+k}^t |\sigma_i| \leq \sum_{i=k}^p |X_i|$,

легко доказываемым по индукции, получаем

$$\begin{aligned} t - (p - k + 1) &\geq \left\lfloor \frac{1}{M} \left(n - \sum_{i=t-p+k}^t |\sigma_i| \right) \right\rfloor \geq \\ &\geq \left\lfloor \frac{1}{M} \left(n - \sum_{i=k}^p |X_i| \right) \right\rfloor = \left\lfloor \frac{1}{M} \sum_{i=1}^{k-1} |X_i| \right\rfloor. \end{aligned}$$

Отсюда $t \geq H$, следовательно, и $\rho_M(G) \geq H$. Отметим, что $\rho_M(G) \leq p - k + 1 + \sum_{i=1}^{k-1} \lfloor |X_i|/M \rfloor$.

Для доказательства формулы (5.19) достаточно теперь показать, что граф G можно разрезать на H кластеров с соблюдением условия теоремы 4. Для этого достаточно, в свою очередь, доказать, что

$$\rho_M(G_{k-1}) = \left\lfloor \frac{1}{M} \sum_{i=1}^{k-1} |X_i| \right\rfloor, \quad (5.20)$$

где G_{k-1} — подграф, порожденный множествами X_1, \dots, X_{k-1} .

Формула (5.20) справедлива, если $k=1$ или $k=2$. Пусть $k \geq 3$. Рассмотрим последовательность подграфов G_1, \dots, G_{k-1} , где G_r — подграф, порожденный мно-

жествами X_1, \dots, X_r . Так как X_1 — внутренне устойчивое множество, т. е. такое множество, никакие две вершины которого не смежны, то

$$\rho_M(G_1) = |X_1|/M.$$

Пусть справедлива формула

$$\rho_M(G_r) = \left\lfloor \frac{1}{M} \sum_{i=1}^r |X_i| \right\rfloor, \quad 1 \leq r < k-1. \quad (5.21)$$

Покажем, что формула (5.21) остается верной при замене r на $r+1$. Это очевидно, если $\sum_{i=1}^r |X_i|$ делится на M . Пусть деление происходит с остатком, равным d , $1 \leq d < M-1$. Можно считать, что для кластеров X_1, \dots, X_{h_r} , $h_r = \rho_M(G_r)$, разрезания графа G_r выполняются равенства

$$|\widetilde{X}_i| = M, \quad i = 1, \dots, h_r-1, \quad |\widetilde{X}_{h_r}| = d.$$

Далее, из условия (5.17) имеем $|X_{r+1}| \geq m(M-1) + 1 = (m-1)(M-1) + M = (m-1)d + M$, т. е. $|X_{r+1}| - md \geq M-d$. Последнее неравенство показывает, что в множестве X_{r+1} найдутся $M-d$ вершин, которые, соблюдая условия теоремы 4, можно поместить в кластер \widetilde{X}_{h_r} . Следовательно, удовлетворяя условиям теоремы 4, подграф G_{r+1} можно разрезать на

$$h_{r+1} = h_r + \left\lfloor \frac{1}{M} (|X_{r+1}| + d - M) \right\rfloor = \left\lfloor \frac{1}{M} \sum_{i=1}^{r+1} |X_i| \right\rfloor$$

кластеров. Ясно, что это разрезание будет минимальным, т. е. $h_{r+1} = \rho_M(G_{r+1})$. Формула (5.20), а вместе с ней и формула (5.19) доказаны.

1.8. Метод ветвей и границ для разрезания бесконечного орграфа. В применении к задаче о разрезании на минимальное число кластеров метод ветвей и границ заключается в пошаговом построении некоторого дерева \mathcal{D} , по которому получается оптимальное разрезание. Каждый шаг построения дерева \mathcal{D} сводится к операции разветвления над множеством разрезов. Опишем эту операцию. Пусть Q — множество разрезов графа G и $k \geq 0$ — такое число, что для любых двух разрезов (X_1, \dots, X_n) и (X'_1, \dots, X'_n) , входящих в Q , $X_i = X'_i$ для всех $i \leq k$.

Построим подграф \bar{G} графа G , удалив из G все вершины, входящие в кластеры X_1, \dots, X_k , и инцидентные им дуги. Пусть Σ — множество выходных вершин подграфа \bar{G} . Для любого разрезания из множества Q его $k+1$ -й кластер является подмножеством множества Σ . Обозначим через ω наименьшее из чисел M и $|\Sigma|$, $\omega = \min\{M, |\Sigma|\}$. Определим теперь систему W_1, \dots, W_r подмножеств множества Σ :

а) если $|\Sigma| \leq M$, то $r = 1$, $W_1 = \Sigma$;

б) если $|\Sigma| > M$, то $r = C_{|\Sigma|}^M$;

W_1, \dots, W_r — всевозможные подмножества множества Σ из M элементов. Очевидно, что $|W_i| = \omega$, $i \in [1:r]$. Обозначим через Q_i множество всех разрезаний из Q с $k+1$ -м кластером W_i , $i \in [1:r]$. Тогда

$$Q = \bigcup_{i=1}^r Q_i \cup Q^*,$$

$$Q_i \cap Q_j = \emptyset, \quad Q_i \cap Q^* = \emptyset, \quad i, j = 1, \dots, r, \quad i \neq j,$$

где Q^* — множество всех тех разрезаний из Q , у которых $k+1$ -й кластер не совпадает ни с одним из множеств W_1, \dots, W_r .

Очевидно, что для любого оптимального разрезания из Q^* можно указать по крайней мере одно разрезание с равным числом кластеров, принадлежащее одному из множеств Q_i , $i \in [1:r]$. Вследствие этого множество Q^* при поиске оптимального разрезания можно не рассматривать.

Исходя из полученного множества разрезаний Q , построим дерево, состоящее из одной звезды, которую будем обозначать через $Z(Q)$. Пусть $Z(Q) = \{(Q, Q_1), \dots, (Q, Q_r)\}$. Операцию, сопоставляющую множеству разрезаний Q звезду $Z(Q)$, будем называть *разветвлением* Q .

Для множеств Q_1, \dots, Q_r нижняя граница числа кластеров разрезаний одинакова и зависит только от Q :

$$\rho_M(Q) = (|\bar{G}| - \omega)/M + k + 1.$$

Рассмотрим набор Q_1, \dots, Q_k непересекающихся множеств разрезаний графа G и набор целых чисел R_1, \dots, R_k , $R_j \geq 0$, таких, что если два разрезания принадлежат одному множеству Q_j , то i -е кластеры этих разрезаний совпадают для всех i , $i \in [1:R_j]$. Предполо-

жим, что для множеств Q_i , $i \in [1:k]$, определена нижняя граница числа кластеров разрезов $\rho_M(Q_i)$. Будем говорить, что $Q_n \leq Q_g$, если либо $\rho_M(Q_n) < \rho_M(Q_g)$, либо $R_n \geq R_g$ в случае равенства чисел $\rho_M(Q_n)$ и $\rho_M(Q_g)$.

Рассмотрим процесс построения дерева \mathcal{D} .

1°. *Базис построения.* Пусть Q_0 — множество всех возможных разрезов графа G с числом элементов в кластерах, не превосходящим M . В качестве базиса построения возьмем дерево \mathcal{D}_0 , состоящее из единственной вершины Q_0 . Это дерево будет наращиваться на следующих шагах процесса. Нижняя граница числа кластеров в разрезах из Q_0 будет

$$\rho_M(Q_0) = \|X\|/M.$$

2°. *$j+1$ -й шаг построения.* Пусть на j -м шаге построено дерево \mathcal{D}_j . Каждая его вершина Q есть множество разрезов графа G . Если вершина Q находится на расстоянии $k > 0$ от корня дерева, то у любых двух разрезов, принадлежащих Q , совпадают их кластеры X_i для $i \in [1:k]$. Для каждой вершины Q определена нижняя граница числа кластеров разрезов, входящих в вершину Q . Упорядочим множество всех висячих вершин дерева \mathcal{D}_j с помощью функции предпочтения. Пусть теперь Q^* — наименьшая из висячих вершин дерева \mathcal{D}_j . При этом определяется нижняя граница числа кластеров разрезов для множеств Q_1, \dots, Q_r , полученных при разветвлении вершины Q^* . Дерево \mathcal{D}_{j+1} получается из дерева \mathcal{D}_j присоединением звезды $Z(Q^*)$.

3°. *Окончание процесса.* Процесс обрывается тогда, когда построена висячая вершина Q , разветвление которой невозможно, т. е. при выполнении шага, на котором $\Sigma = \bar{G}$. Полученное в результате дерево \mathcal{D} есть искомое дерево. Очевидно, множество Q состоит из единственного разреза, которое и представляет собой оптимальное разрезание.

Несколько слов о представлении деревьев \mathcal{D}_j . Каждая вершина Q , находящаяся на расстоянии R от корня дерева \mathcal{D}_j , представляет собой множество разрезов с зафиксированными первыми R кластерами. Обозначим эти зафиксированные кластеры через X_1, \dots, X_R . Будем идентифицировать вершину Q в \mathcal{D}_j списком вершин кластера X_R и порядковым номером

появления Q в процессе построения. Порядковый номер необходим, так как X_R может быть кластером разрезания, не входящего в Q . При таком способе задания вершин по набору дуг дерева \mathcal{D}_i однозначно восстанавливаются вершины дерева, как множества разрезов. При представлении \mathcal{D}_i целесообразно делить его на каждом шаге процесса на две части. Первая часть — это дерево $\bar{\mathcal{D}}_i$, состоящее из множества звезд \mathcal{D}_{ij} , за исключением висячих звезд, т. е. звезд, дуги которых инцидентны висячим вершинам. Вторая часть — это множество висячих звезд вместе с информацией о границах и о расстоянии от вершин до корня, упорядоченное с помощью функции предпочтения. Такой способ представления дерева приводит к сокращению объема информации и к сокращению времени поиска разветвляемой вершины.

§ 2. Раскраска графов

2.1. Правильная раскраска. Определение хроматического числа. *Правильной k -раскраской* графа $G = (X, E)$ в k цветов называется такое разбиение множества X его вершин на k непересекающихся подмножеств X_1, \dots, X_k , что никакие две вершины из одного и того же подмножества не смежны.

Множество вершин одного цвета, а это в точности вершины одного из подмножеств X_1, \dots, X_k , называется *одноцветным классом*. Итак, при правильной k -раскраске множество вершин графа разбивается на k одноцветных классов.

Хроматическим числом $\chi(G)$ графа G называется наименьшее k , при котором G имеет правильную k -раскраску.

Будем говорить, что граф G *k -раскрашиваем*, если $\chi(G) \leq k$, и *k -хроматичен*, если $\chi(G) = k$. Если n есть число вершин графа, то граф, очевидно, имеет n -раскраску и $\chi(G)$ -раскраску и, кроме того, имеет k -раскраску для любого k , удовлетворяющего неравенству $\chi(G) \leq k \leq n$. Легко убедиться, что хроматическое число полного n -вершинника равно n , пустого графа, т. е. графа, не содержащего ни одного ребра, равно единице, четного цикла равно двум, нечетного цикла — трем. Очевидно, что граф является 1-хроматическим тогда и только тогда, когда он пуст.

Теорема 5. *Граф бихроматичен (2-хроматичен) тогда и только тогда, когда он не содержит нечетных простых циклов.*

Следствие 1. *Любое дерево бихроматично.*

Следствие 2. *Любой двудольный граф бихроматичен.*

Характеризация k -хроматических графов для $k \geq 3$ не известна. Поэтому в теории раскрасок графов наибольший интерес представляют оценки хроматического числа и приближенные методы раскраски.

2.2. Некоторые оценки хроматического числа графов. Очевидно, что если граф G содержит в качестве своего подграфа полный t -вершинник, то его хроматическое число удовлетворяет неравенству $t \leq \chi(G) \leq n$, где n — число вершин. Большую информацию можно получить, если привлечь дополнительные характеристики графа.

Лемма 5. *Если наибольшая из степеней вершин графа G равна ρ , то этот граф $\rho + 1$ -раскрашиваем.*

Доказательство проводится индукцией по числу вершин.

Теорема 6. *Пусть наибольшая из степеней вершин графа G равна ρ . Тогда G является ρ -раскрашиваемым, за исключением тех случаев, когда граф G содержит в качестве компоненты полный $\rho + 1$ -вершинник или когда $\rho = 2$ и одна из компонент графа G является нечетным циклом.*

Лемму 5 и теорему 6 удобно применять тогда, когда степени вершин примерно одинаковы. Если же граф имеет вершины с резко отличающимися степенями, в особенности если вершин с большой степенью немного, они дают чрезвычайно завышенные оценки; так, звездный граф, имеющий одну вершину степени ρ , а остальные вершины степени 1, по теореме 6 является ρ -раскрашиваемым, а на самом деле он бихроматичен.

Теорема 7. *Для любого графа G*

$$\chi(G) \leq 1 + \max_{\bar{H} \subset G} \min_{v \in H} \deg_H v,$$

где максимум берется по всем собственным подграфам \bar{H} графа G .

Доказательство. Утверждение очевидно для пустых графов. Пусть G — произвольный h -хроматический граф, $h \geq 2$, H — любой наименьший подграф,

у которого $\chi(H) = h$. Тогда $\chi(H \setminus v) = h - 1$ для всех вершин v графа H . Следовательно, $\deg v \geq h - 1$, так что $\min_{v \in H} \deg v \geq h - 1$, и потому

$$h - 1 \leq \min_{v \in H} \deg v \leq \max_{H'} \min_{v \in H'} \deg v \leq \max_{\bar{H}} \min_{v \in \bar{H}} \deg v,$$

где первый максимум берется по всем подграфам H' графа H , второй — по всем подграфам \bar{H} графа G . Отсюда вытекает, что

$$\chi(G) = h \leq 1 + \max_{\bar{H}} \min_{v \in \bar{H}} \deg v.$$

Теорема 8. Для любого графа G справедлива оценка

$$n/\beta_0 \leq \chi(G) \leq n - \beta_0 + 1,$$

где n — число вершин в G , β_0 — вершинное число независимости.

Доказательство. Если $\chi(G) = k$, то множество X вершин можно разбить на k одноцветных классов X_1, \dots, X_k , каждый из которых является независимым множеством, т. е. множеством, в котором никакие две вершины не смежны. Если $|X_i| = n_i$, то $n_i \leq \beta_0$ для всех i , так что $n = \sum_i n_i \leq k\beta_0$.

Для проверки верхней оценки рассмотрим максимальное независимое множество S , содержащее β_0 вершин. Ясно, что $\chi(G \setminus S) \geq \chi(G) - 1$. Так как $G \setminus S$ имеет $n - \beta_0$ вершин, то $\chi(G \setminus S) \leq n - \beta_0$. Отсюда $\chi(G) \leq \chi(G \setminus S) + 1 \leq n - \beta_0 + 1$.

Теорема 9. Для любого графа G справедлива оценка

$$\begin{aligned} - \left| \left[- \frac{n}{\left[\frac{n^2 - 2m}{n} \right]} \left(1 - \frac{\{(n^2 - 2m)/n\}}{1 + \lfloor (n^2 - 2m)/n \rfloor} \right) \right] \right| &\leq \\ &\leq \chi(G) \leq \left\lceil \frac{3 + \sqrt{9 + 8(m - n)}}{2} \right\rceil, \end{aligned}$$

где n — число вершин, m — число ребер.

(Скобки $[]$ обозначают целую часть, а скобки $\{ \}$ — дробную часть числа.)

Доказательство теоремы будет дано ниже.

Приведем еще оценку, вывод которой будет рассмотрен при изучении приближенных методов раскраски графа.

Теорема 10. Пусть в графе G вершины упорядочены по убыванию степеней. Тогда $\chi(G) \leq \max_{1 \leq i \leq n} \min\{i, 1 + \deg v_i\}$.

2.3. Говорят, что граф укладывается на поверхности S , если его можно так нарисовать на S , что никакие два ребра не пересекаются в точках, отличных от концов. Граф называется *планарным* или *топологически плоским* графом, если его можно уложить на плоскости; *плоский* граф — это граф, уже уложенный на плоскости.

Плоскость, на которую уложен граф, разбивается на ряд областей; области, ограниченные со всех сторон ребрами графа, называются его *внутренними гранями* или просто *гранями*; неограниченную область называют *внешней гранью*. Важнейшим результатом, касающимся плоских графов и лежащим в основе многих исследований, является следующая теорема (или формула) Эйлера.

Теорема 11. Пусть G — связный плоский граф, и пусть n , m и f обозначают соответственно число вершин, ребер и граней графа G (включая внешнюю грань). Тогда $n + f = m + 2$.

Следствие 1. Если G — связный простой, т. е. без петель и кратных ребер, планарный граф с n вершинами, $n \geq 3$, и m ребрами, то $m \leq 3n - 6$.

Так как каждая грань ограничена при наших предположениях по крайней мере тремя ребрами, то при подсчете числа ребер вокруг каждой грани имеем $3f \leq 2m$; применяя формулу Эйлера, получаем требуемое.

Следствие 2. В любом планарном графе без петель и кратных ребер существует вершина, степень которой не больше пяти.

В самом деле, пусть в графе по меньшей мере три вершины. Если степень каждой вершины не меньше шести, то имеет место неравенство $6n \leq 2m$, или $3n \leq m$. Из следствия 1 сразу же вытекает, что $3n \leq 3n - 6$ — очевидное противоречие.

Теорема 12. Каждый планарный граф 5-раскрашиваем.

Доказательство теоремы ведется индукцией по числу вершин. Для любого планарного графа с $n \leq 5$ вершинами результат тривиален, так как любой граф n -раскрашиваем. Допустим, что все планарные графы с n вершинами, $n \geq 5$, 5-раскрашиваемы. Пусть G —

планарный граф с $n + 1$ вершинами. В силу следствия 2 из теоремы 11 в графе G найдется вершина v степени пять или менее. По индуктивному предположению планарный граф $G \setminus v$ 5-раскрашиваем. Рассмотрим некоторую 5-раскраску графа $G \setminus v$ цветами c_1, \dots, c_5 . Ясно, что если некоторый цвет, скажем c_3 , не используется в раскраске вершин, смежных с вершиной v , то, приписав цвет c_3 вершине v , получим 5-раскраску графа G .

Осталось рассмотреть случай, когда степень вершины v равна пяти и для вершин графа G , смежных с вершиной v , используются все пять цветов. Без ограничения общности можно считать граф G плоским, так что мы можем говорить о циклической упорядоченности смежных с v вершин. Переставим номера цветов, если это необходимо, так, чтобы цвета вершин,

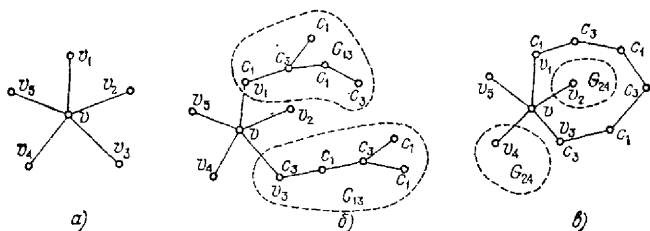


Рис. 5.13.

смежных с v , при обходе их по часовой стрелке были c_1, c_2, c_3, c_4, c_5 . Обозначим теперь вершину, смежную с вершиной v и окрашенную в цвет c_i , через v_i , $1 \leq i \leq 5$ (рис. 5.13, а). Обозначим через $G_{\alpha\beta}$ подграф графа $G \setminus v$, порожденный всеми вершинами, окрашенными в цвета α и β . Возьмем теперь пару не расположенных рядом вершин, смежных с вершиной v , например v_1 и v_3 . Если эти вершины принадлежат различным компонентам связности подграфа G_{13} (рис. 5.13, б), то 5-раскраску графа $G \setminus v$, при которой вершина v не имеет смежных вершин, окрашенных в цвет c_3 , можно получить, поменяв в той компоненте подграфа G_{13} , которая содержит вершину v_3 , цвета вершин с c_3 на c_1 и обратно (перекраска вершин). После этого, окрасив вершину v в цвет c_3 , образуем 5-раскраску графа G .

Если же вершины v_1 и v_3 принадлежат одной и той же компоненте связности подграфа G_{13} , то в графе

$G \setminus v$ между вершинами v_1 и v_3 существует простая цепь, все вершины которой окрашены в цвета c_1 и c_3 (рис. 5.13, в). Эта цепь вместе с цепью $\{v_1, v_2, v_3\}$, образует простой цикл, который окружает либо вершину v_2 , либо вершины v_4 и v_5 . В любом из этих случаев v_2 и v_4 нельзя соединить простой цепью, все вершины которой окрашены в цвета c_2 и c_4 . Следовательно, рассматривая подграф G_{24} графа $G \setminus v$, заключаем, что вершины v_2 и v_4 принадлежат различным его компонентам связности. Таким образом, если поменять между собой цвета вершин в компоненте подграфа G_{24} , содержащей вершину v_2 , то получится 5-раскраска графа $G \setminus v$, в которой ни одна из вершин, смежных с вершиной v , не будет окрашена в цвет c_2 . Поэтому, окрасив вершину v в цвет c_2 , образуем 5-раскраску исходного графа.

2.4. Приближенные методы раскраски графов. Для упорядоченного множества вершин v_1, \dots, v_n графа G *последовательной раскраской*, отвечающей этому порядку, называется раскраска, использующая каждый из цветов c_1, \dots, c_k и определяемая следующим образом:

- а) вершине v_1 приписан цвет c_1 ;
- б) если подграф $H_{\langle v_1, \dots, v_{i-1} \rangle}$, порожденный вершинами v_1, \dots, v_{i-1} , k' -раскрашен, $k' \leq i-1$, то вершина v_i получает цвет c_m , где $m \leq k' + 1$, т. е. цвет с наименьшим номером, не встречающийся на смежных с вершиной v_i вершинах.

Заметим, что число k при таком способе раскраски заранее не фиксируется. Выбор данного способа раскраски в качестве приближенного метода основан на том, что этот простой способ дает точную k -раскраску для одного частного вида графов — полных k -дольных графов. Применение его к графам произвольного вида дает, естественно, приближенную к оптимальной раскраску.

k -дольным называется граф, множество вершин которого можно разбить на k непересекающихся подмножеств X_1, \dots, X_k так, что никакие две вершины из подмножества X_i , $i \in [1:k]$, не смежны. k -дольный граф есть обобщение двудольного графа. k -дольный граф называется *полным k -дольным*, если каждая вершина из множества X_i , $i \in [1:k]$, смежна с каждой вершиной из множества X_j , $j \neq i$.

Лемма 6. *Любая последовательная раскраска полного k -дольного графа есть k -раскраска.*

Пусть $G = (V, E)$ — полный k -дольный граф. Если $v \in V_i$ и $w \in V_j$, $i \neq j$, то вершины v и w должны быть окрашены в разные цвета при любой последовательной раскраске, поскольку вершины v и w смежны в графе G . Пусть v_1, \dots, v_n — фиксированное упорядочение множества вершин V , и пусть $v_i, v_j \in V_p$, $i < j$; тогда у вершины v_j в подграфе $H_{\langle v_1, \dots, v_j \rangle}$ те же соседи, что и у вершины v_i . Поэтому наименьший номер цвета, отсутствующего на смежных с v_j вершинах в подграфе $H_{\langle v_1, \dots, v_{j-1} \rangle}$, должен быть номером цвета вершины v_i . Следовательно, вершины v_i и v_j будут окрашены алгоритмом, отвечающим этому упорядочению, одинаково. В итоге все вершины каждого из множеств V_i , $i \in [1:k]$, имеют один цвет, и граф G k -раскрашен. Поскольку упорядочение вершин произвольно, то лемма доказана.

Последовательные раскраски в общем случае не обеспечивают раскраску графа в минимальное число цветов.

Перейдем к рассмотрению различных стратегий при организации последовательных раскрасок. Теорема 6, давая оценку хроматического числа через наибольшую степень, приводит к мысли начать раскрашивать с вершин большой степени. Если таких вершин мало, то можно обойтись меньшим числом цветов. Упорядочим вершины графа в порядке убывания их степеней и применим описанную выше последовательную раскраску. После того как будет окрашено i вершин, будет израсходовано самое большее i цветов, а потребуются для окраски оставшихся не более $1 + \max_i \deg v_i$ цветов (лемма 5), откуда и получаем оценку, даваемую теоремой 10:

$$\chi(G) \leq \max_{1 \leq i \leq n} \min \{i, 1 + \deg v_i\}.$$

Эта оценка называется *оценкой хроматического числа через усеченную максимальную степень*.

Упорядочение вершин в соответствии с убыванием их степеней назовем НП-упорядочением («наибольшие — первыми»), а алгоритмы, основанные на таком упорядочении, назовем НП-алгоритмами. Заметим, что требуемое НП-алгоритмом число цветов может варьироваться из-за неоднозначности НП-упорядочения, но никогда не превосходит числа, определяемого оценкой через усеченную максимальную степень.

Более детальное рассмотрение процедуры последовательной раскраски показывает, что для заданного упорядочения v_1, \dots, v_n вершин графа G соответствующий алгоритм последовательной раскраски требует не более k красок, где

$$k = \max_{1 \leq i \leq n} \{1 + \deg_{H_{\langle v_1, \dots, v_i \rangle}} v_i\}. \quad (5.22)$$

Здесь $\deg_{H_{\langle v_1, \dots, v_i \rangle}} v_i$ обозначает степень вершины v_i в подграфе $H_{\langle v_1, \dots, v_i \rangle}$.

Упорядочение вершин, минимизирующее k в (5.22), строится следующим образом:

а) для $n = |V|$ в качестве v_n выбирается вершина минимальной степени в графе G ;

б) для $i = n-1, n-2, \dots, 2, 1$ в качестве v_i выбирается вершина минимальной степени в подграфе $H_{\langle V \setminus \{v_n, \dots, v_{i+1}\} \rangle}$.

Упорядочение вершин такого вида будем называть ПН-упорядочением («последними — наименьшие»). Следует отметить, что, во-первых, произвольное ПН-упорядочение минимизирует k в (5.22) по всем $n!$ возможным упорядочениям, во-вторых, определение ПН-упорядочения носит характер рекурсивности, не присутствующей в процедуре НП-упорядочения. Дело в том, что степени вершин, вычисляемые при ПН-упорядочении, относятся к подграфам, тогда как при НП-упорядочении степени вершин вычисляются в самом графе. Таким образом, эти упорядочения не эквивалентны.

Процедура определения ПН-упорядочения вершин и нахождения по нему раскраски будет называться ПН-алгоритмом. По своему строению ПН-алгоритм приводит к раскраске не более чем $1 + \max_{H \subset G} \min_{v \in H} \deg_H v$ цветами, что в точности совпадает с оценкой, даваемой теоремой 7. Эта оценка называется *максимальной по подграфам через минимальную степень* оценкой хроматического числа. Очевидно, что она или точнее, или совпадает с оценкой через усеченную максимальную степень.

Для улучшения качества раскраски, даваемой ПН-алгоритмом, к k -раскраске может быть применена дополнительная процедура перекраски. В самом деле, когда вершина v_i присоединяется к $k-1$ -раскрашенному подграфу $H_{\langle v_1, \dots, v_{i-1} \rangle}$, то k -й цвет требуется

лишь в том случае, если вершина v_i смежна с вершинами, окрашенными в цвета c_1, \dots, c_{k-1} . Если эти вершины образуют полный $k-1$ -вершинник, то необходим цвет c_k . В противном случае возможна такая перекраска некоторых смежных с v_i вершин, что подграф $H_{\langle v_1, \dots, v_{i-1} \rangle}$ будет по-прежнему окрашен $k-1$ цветами, но на вершины, смежные с вершиной v_i , будет потрачено не более $k-2$ цветов.

2.5. Алгоритм с перекраской двуцветных подграфов — ПНП-алгоритм.

Вход: Граф G с ПН-упорядоченными вершинами.

Выход: Субоптимальная раскраска вершин.

начало

1. $j := 1$;

2. для i от 1 до n шаг 1 цикл

3. **начало**

4. $m :=$ наименьший номер цвета, отсутствующего на вершинах, смежных с вершиной v_i ;

5. **если** $m < j$ **то**

6. окрасить вершину v_i в цвет c_m ;

7. **иначе начало**

8. $K :=$ множество цветов, представленных ровно один раз на вершинах, смежных с вершиной v_i ;

9. **если** найдется пара $\alpha, \beta \in K$, такая, что вершины v_α и v_β , смежные с v_i и окрашенные в цвета α и β , не соединены двуцветной цепью **то**

10. **начало**

11. перекрасить ту компоненту двуцветного графа $G_{\alpha\beta}$, которая содержит вершину v_α ;

12. окрасить вершину v_i в цвет α ;

13. **конец**

14. **иначе начало**

15. $j := j + 1$;

16. окрасить вершину v_i в цвет c_j ;

17. **конец**

18. **конец**

19. **конец цикла**;

конец

Теорема 13. Алгоритм последовательной раскраски вершин, основанный на ПН-упорядочении вершин с пошаговой перекраской, требует не более 5 цветов для раскраски любого планарного графа.

Доказательство будем вести индукцией по числу вершин i . Пусть вершины планарного или даже пло-

ского (для определенности) графа G ПН-упорядочены, т. е. вершина v_i имеет минимальную степень в графе $H_{\langle v_1, \dots, v_i \rangle}$, $i \in [1 : n]$. Для $i = 1$ граф $H_{\langle v_1 \rangle}$ 1-раскрашен. Предположим, что для некоторого i граф $H_{\langle v_1, \dots, v_{i-1} \rangle}$ 5-раскрашен ПНП-алгоритмом. Поскольку граф $H_{\langle v_1, \dots, v_i \rangle}$ плоский, то степень вершины v_i в графе $H_{\langle v_1, \dots, v_i \rangle}$ не превосходит пяти. Если на вершинах, смежных с вершиной v_i в подграфе $H_{\langle v_1, \dots, v_i \rangle}$, встречаются не все пять цветов, то вершина v_i будет окрашена ПНП-алгоритмом без обращения к шестому цвету. Если, напротив, каждый цвет представлен по одному разу, то существует некоторая перекраска, приводящая к уменьшению числа цветов на вершинах, смежных с вершиной v_i . Поскольку ПНП-алгоритм просматривает все возможные перекраски, то подходящая перекраска будет найдена и подграф $H_{\langle v_1, \dots, v_i \rangle}$ будет раскрашен пятью цветами. Теорема доказана.

Если граф G не содержит подграфа с минимальной степенью 5 или более, то для ПН-упорядочения вершин

$$\deg_{\langle v_1, \dots, v_i \rangle} v_i = \min_{1 \leq j < i} \deg_{\langle v_1, \dots, v_i \rangle} v_j \leq 4,$$

и можно показать, что ПНП-алгоритм в этом случае раскрашивает граф в четыре цвета.

2.6. Алгоритм Д. Рассмотрим более эффективный алгоритм последовательной раскраски. Назовем *степенью насыщения* вершины v в частично раскрашенном графе число различных цветов на смежных с v вершинах. В алгоритме Д последовательно окрашиваются вершины с наибольшей степенью насыщения.

А л г о р и т м:

начало

1. упорядочить вершины в порядке невозрастания их степеней (НП-упорядочение);

2. вершину v_1 окрасить в цвет c_1 ;

3. для i от 1 до $n - 1$ шаг 1 цикл

4. **начало**

примечание в графе окрашено i вершин **конец**
примечания

5. сформировать множество K вершин с наибольшей степенью насыщения;

6. выбрать из K вершину v_{i+1} с наибольшей степенью в нераскрашенном подграфе;

7. окрасить v_{i+1} в цвет с наименьшим возможным номером;

8. конец цикла;

конец

2.7. Алгоритм Ершова — Кожухина. В основе этого алгоритма лежит принцип склеивания вершин и сведение исходного графа с помощью такого склеивания к графу с меньшим числом вершин.

Назовем *окрестностью 1-го порядка* $R_1(v)$ вершины v множество вершин, смежных с вершиной v . *Склеивание* вершин v_1 и v_2 с окрестностями $R_1(v_1)$ и $R_2(v_2)$ соответственно есть преобразование, состоящее в удалении вершин v_1 и v_2 и добавлении новой вершины v , смежной со всеми теми вершинами, с которыми были смежны v_1 и v_2 , так что $R_1(v) = R_1(v_1) \cup R_1(v_2)$. Это не приводит к появлению параллельных ребер, так как вершины удаляются вместе с инцидентными ребрами. В результате склеивания пары вершин получается граф с числом вершин, на единицу меньшим, а также, возможно, с меньшим числом ребер. Две вершины графа называются *соцветными*, если существует минимальная раскраска вершин графа, в которой они раскрашены в один цвет.

Лемма 7. *В любом графе, отличном от полного, существует по крайней мере одна пара соцветных вершин.*

Доказательство. Пусть граф G имеет n вершин. Если ни в одной минимальной раскраске нет двух вершин, окрашенных в один цвет, то $\chi(G) = n$. Так как G — неполный граф, то в нем есть хотя бы одна пара несмежных вершин v_1 и v_2 . Склеим эти две вершины. Получившийся граф G' будет иметь $n - 1$ вершин, а стало быть, $\chi(G') \leq n - 1$. Взяв любую раскраску графа G' в $n - 1$ цвет (или менее), найдем в ней вершину, в которую были склеены вершины v_1 и v_2 . Расклеив эти вершины и сохранив их цвет в данной раскраске графа G' , получим раскраску графа G в $n - 1$ или менее цветов, что противоречит предположению $\chi(G) = n$.

Лемма 8. *Если граф G' получается из графа G склеиванием пары соцветных вершин, то $\chi(G') = \chi(G)$.*

Теорема 14. *Для всякого графа с хроматическим числом χ существует последовательность попарных*

склеиваний вершин, приводящая к полному χ -вершиннику.

Доказательство. Указанная последовательность склеиваний строится следующим образом. В исходном графе ищем пару соцветных вершин и склеиваем. В силу леммы 8 хроматическое число при этом остается неизменным. В силу леммы 7 такое склеивание можно делать до тех пор, пока не будет получен полный граф, в котором по лемме 8 не может быть более $\chi(G)$ вершин. Теорема доказана.

Практически в доказательстве теоремы 14 содержится алгоритм раскраски, исключая момент выбора соцветных вершин. Так как вся сложность решения задачи о раскраске указанным методом заключается в построении последовательности соцветных вершин, то необходима подходящая эвристика для этого шага.

Теорема 15. Для любой вершины v , у которой окрестность 1-го порядка $R_1(v)$ не совпадает со всем множеством $X \setminus \{v\}$, в окрестности 2-го порядка $R_2(v)$ имеется по крайней мере одна соцветная вершина.

Доказательство. Рассмотрим некоторую минимальную раскраску графа G , в которой вершина v получает цвет α . Если в этой раскраске некоторая вершина w из $R_2(v)$ будет также окрашена в цвет α , то все доказано (рис. 5.14, а).

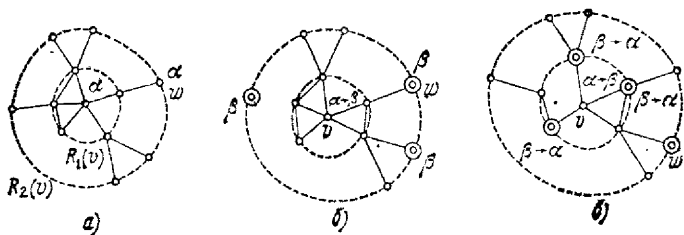


Рис. 5.14.

Рассмотрим теперь случай, когда ни одна вершина из $R_2(v)$ не окрашена в цвет α (рис. 5.14, б). Возьмем в этом случае любую вершину w из $R_2(v)$, окрашенную в цвет β , и рассмотрим окрестность $R_1(v)$ вершины v . Возможны два случая: в $R_1(v)$ нет вершин, окрашенных в цвет β , и в $R_1(v)$ имеются вершины, окрашенные в цвет β . В первом случае перекрасим вершину v в цвет β , и теорема в этом случае доказана. Во втором случае обозначим множество вершин в ок-

рестности $R_1(v)$, окрашенных в цвет β , через R' . Так как для любой вершины $u \in R_1(v)$ справедливо включение $R_1(u) \subseteq \{v\} \cup R_1(v) \cup R_2(v)$ (рис. 5.14, в), то ясно, что для любой вершины из R' вершина v является ее единственной смежной вершиной, окрашенной в цвет α . Перекрасив теперь вершины из R' в цвет α , убрав предварительно его с вершины v , получим, что цвет β исчез из окрестности $R_1(v)$. Приписав вершине v цвет β , докажем справедливость теоремы и в этом случае.

Хотя алгоритм Ершова — Кожухина, основанный на склеивании соцветных вершин, несколько уступает по затратам времени алгоритмам последовательной раскраски, он обладает одним значительным преимуществом — он дает возможность работать с большими графами. Эвристика, определяемая теоремой 15, может быть усложнена выбором в качестве кандидатов на склейку той пары вершин, у которых наибольшее число общих смежных вершин.

2.8. Доказательство теоремы 9. Докажем верхнюю оценку, т. е. неравенство

$$\chi(G) \leq [(3 + \sqrt{9 + 8(m - n)})/2].$$

Пусть граф G имеет n вершин и m ребер. Если граф G не полный, то в нем найдется пара соцветных вершин, склеивая которые мы получаем граф с тем же хроматическим числом, но в котором вершин на единицу меньше и по крайней мере на одно ребро меньше, чем в исходном. Пусть после s склеек ($s < n$) мы получили полный $n - s$ -вершинник G_s , для которого

$$\chi(G_s) = \chi(G) = n - s. \quad (5.23)$$

Граф G_s будет иметь $\chi(G)(\chi(G) - 1)/2$ ребер, т. е. будет иметь на $m - \chi(G)(\chi(G) - 1)/2$ ребер меньше, чем граф G . Но поскольку на каждом i -м шаге, $i \in [1 : s]$, граф G_i содержал по крайней мере на одно ребро меньше, чем граф G_{i-1} , ($G_0 = G$), то

$$m - \chi(G)(\chi(G) - 1)/2 \geq s$$

или, в силу (5.23),

$$m - (\chi(G)(\chi(G) - 1))/2 \geq n - \chi(G). \quad (5.24)$$

Решая это неравенство относительно $\chi(G)$, получим

$$\chi(G) \leq (3 + \sqrt{9 + 8(m - n)})/2 \quad (5.25)$$

для любого графа G . Найдем теперь граф G' с n вер-

пинами и m ребрами, для которого $\chi(G') = \kappa$, где

$$\kappa = \lfloor (3 + \sqrt{9 + 8(m - n)})/2 \rfloor. \quad (5.26)$$

Нетрудно увидеть, что число ребер в связном графе удовлетворяет неравенствам

$$n - 1 \leq m \leq n(n - 1)/2.$$

Отсюда и из (5.26) вытекает, что

$$\kappa \leq n, \quad \kappa(\kappa - 1)/2 \leq m, \quad n - \kappa \leq m - \kappa(\kappa - 1)/2. \quad (5.27)$$

Граф G' строится следующим образом: вначале строится полный κ -вершинник, к некоторой произвольной его вершине присоединяется цепь, содержащая $n - \kappa$ вершин и $n - \kappa$ ребер, а оставшиеся $m - \kappa(\kappa - 1)/2 - (n - \kappa)$ ребер расставляются произвольным образом. Поскольку граф G' содержит в качестве подграфа полный κ -вершинник, то $\chi(G') \geq \kappa$, откуда, в силу (5.25), следует оценка

$$\chi(G') = \lfloor (3 + \sqrt{9 + 8(m - n)})/2 \rfloor.$$

Доказательство нижней оценки. Пусть для графа G с n вершинами и m ребрами $\chi(G) = k$. Назовем *отсутствующим ребром* (v_q, v_r) в графе G любую пару несмежных вершин v_q и v_r , а также все пары вида (v_q, v_q) . Очевидно, что количество отсутствующих ребер в графе G равно $n^2 - 2m$. Рассмотрим любую минимальную раскраску графа G . Пусть X_i — множество вершин, окрашенных в цвет c_i , $i \in [1 : k]$, и $|X_i| = n_i$. Поскольку все вершины из X_i попарно несмежны, то общее количество отсутствующих ребер, образованных только вершинами из X_i , равно n_i^2 . Отсюда сразу же вытекает неравенство

$$\sum_{i=1}^k n_i^2 \leq n^2 - 2m \quad (5.28)$$

при условии, что

$$\sum_{i=1}^k n_i = n. \quad (5.29)$$

Найдем минимум p функции $F = \sum_{i=1}^k n_i^2$ при условии (5.29) на множестве целых чисел. Положим

$$n_i = \lfloor n/k \rfloor + y_i, \quad l = n - \lfloor n/k \rfloor k, \quad (5.30)$$

где n/k — значения n_i , минимизирующие F при условии (5.29) на множестве рациональных чисел. Очевидно, что

$$\sum_{i=1}^k y_i = l, \quad 0 \leq l \leq k. \quad (5.31)$$

Легко показать, что нахождение минимума функции F сводится к нахождению минимума $\sum_{i=1}^k y_i^2$ при условии (5.31). Этот минимум достигается, например, при $y_1 = \dots = y_{k-l} = 0, y_{k-l+1} = \dots = y_k = 1$. Отсюда следует, что

$$p = n^2/k + k(n/k - [n/k])(1 + [n/k] - n/k), \quad (5.32)$$

где n^2/k — минимум F на множестве рациональных чисел. Теперь (5.28) можно переписать в виде

$$k^2(n/k - [n/k])(1 + [n/k] - n/k) \leq (n^2 - 2m)k - n^2. \quad (5.33)$$

Из этого неравенства следует, что

$$\chi(G) \geq v = -[-x_0] \quad (5.34)$$

для любого графа G с n вершинами и m ребрами, где x_0 — корень уравнения

$$x^2(n/x - [n/x])(1 + [n/x] - n/x) = (n^2 - 2m)x - n^2, \quad (5.35)$$

лежащий в отрезке от 1 до n .

Исследуя поведение левой и правой частей уравнения (5.35), легко показать, что $[n/x_0] = [n/x_1]$, где x_1 есть корень уравнения $(n^2 - 2m)x - n^2 = 0$. Вычислив $[n/x_1]$ и подставив его в (5.35), получим линейное уравнение относительно x , из которого находим x_0 :

$$x_0 = \frac{n}{\left[\frac{n^2 - 2m}{n} \right]} \left(\frac{\left\{ \frac{n^2 - 2m}{n} \right\}}{1 + \left[\frac{n^2 - 2m}{n} \right]} \right).$$

Отсюда непосредственно следует справедливость нижней оценки.

Библиографический комментарий

Постановка задач о разрезании графов дана в соответствии с работами [15, 68]. Разрезания деревьев и графов методом динамического программирования исследовались в работах Дж. Лукеса [127, 128]. Алгоритм субоптимального разрезания

графов принадлежит А. С. Никитину и В. Н. Венгерову [62]. Метод разрезания бесконтурных орграфов на минимальное число кластеров принадлежит Г. А. Бекишеву и излагается по работе [8]. Метод ветвей и границ для разрезания бесконтурных орграфов принадлежит Т. А. Тушкиной и К. В. Шахбазян [74].

С правильными раскрасками графов можно познакомиться в книгах [33, 75, 80] и др., там же можно найти ссылки на соответствующую литературу. Заметим, что теорема 5 носит название теоремы Кёнига, теорема 6 — теоремы Брукса, теорема 7 — теоремы Секереша — Вилфа, теорема 9 — теоремы Ершова — Кожухина, теорема 10 — теоремы Уэлша — Пауэла, теорема 12 — теоремы Хивуда. Приближенные методы раскраски излагаются по обзору [130] (см. также [25]), алгоритм Д описан в работе [104] (алгоритм DSATUR), описание алгоритма Ершова — Кожухина дано по книге А. П. Ершова [30], доказательство теоремы 9 — по работе [31]. В алгоритме А. А. Бульонковой [12] сочетаются преимущества алгоритма Ершова — Кожухина с быстродействием алгоритма Д; алгоритм Д дополняется процедурой склеивания одинаково окрашенных вершин на каждом шаге приписывания цвета вершине. В отличие от ее алгоритма, в работе [109] предлагается вычислять для всех пар несмежных вершин число общих смежных вершин и выбирать для склейки пару с наибольшим таким числом. Хорошие обзоры приближенных методов раскраски даны также в [124, 126]. Интерес к этим методам вызывается весьма вероятным отсутствием полиномиального алгоритма (подробнее об этом см. [115]).

ГЛАВА 6

ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ

§ 1. Анализ и тестирование программ. Вычисление характеристик программ

При конструировании и отладке программ возникают задачи, либо сводящиеся к задачам теории графов, либо использующие таковые в качестве основы для решения. К ним в первую очередь относятся задачи анализа потока управления в программе, задачи тестирования и проверки правильности программы, оценки сложности и времени исполнения. Связь между такими задачами и задачами теории графов показана в виде таблицы.

1.1. Пути i -го уровня в управляющих графах и оценка времени исполнения программ. Пути i -го уровня отражают глубину вложенности оператора; чем больше уровень вложенности, тем труднее достичь этого оператора и, значит, тем сложнее проверка правильности его работы. Путь уровня 0 есть простой $s-t$ -путь; путь уровня 1 есть простой путь или контур, который начинается или заканчивается в вершинах путей уровня 0. Аналогично, путь уровня i есть простой путь или контур, который начинается или заканчивается в вершинах путей меньшего уровня и у которого ни одна другая вершина или дуга не входит в путь меньшего уровня. Путь уровня i сопоставляется самому меньшему из возможных уровней.

Задача системного программирования	Соответствующая задача теории графов
Оптимизация программ, в том числе выделение фрагментов (структуризация) и декомпозиция	Нахождение доминаторов Перечисление бикомпонент Перечисление контуров Перечисление интервалов Перечисление гамаков Перечисление линейных компонент Нахождение вложенных зон Нахождение вложенных контуров Интервальное представление графа
Проверка правильности программ	Нахождение транзитивного и обратного транзитивного замыкания Нахождение замыкания относительно множества вершин Перечисление путей Перечисление контуров
Определение порядка обработки операторов при потоковом анализе	Нумерации вершин Построение длинных последовательностей вершин
Тестирование программ	Нахождение покрытия вершин путями Нахождение покрытия дуг путями Нахождение покрытия требуемых путей путями Нахождение кратчайшего пути при наличии дополнительных ограничений Нахождение множества фундаментальных циклов Определение цикломатического числа

В качестве примера приведем полный список путей i -го уровня управляющего графа программы из п. 2.1 главы 1:

Уровень	Пути
0	$\mu[v_1, v_{16}] = [a, b, d]$
1	$\mu[v_2, v_3] = [c, e, f, u, w]$ $\mu[v_2, v_3] = [c, e, g, h, j, u, w]$ $\mu[v_2, v_3] = [c, e, g, h, k, m, u, v]$
2	$\mu[v_8, v_7] = [n, p, q, r, s, t]$
3	нет

Для оценки времени исполнения рассмотрим марковскую граф-модель программы. Будем считать, что с каждой вершиной v граф-модели связан параметр t_v , задающий время исполнения соответствующего оператора. Время исполнения программы есть случайная величина, определяемая параметрами граф-модели и ее структурой. Ее можно интерпретировать как время блуждания точки по графу с заданными временами задержек точки в вершинах и с заданными вероятностями переходов. Траектория движения такой точки есть путь из входа s в выход t графа. Поэтому вычисление оценки сводится к генерированию всевозможных путей из s в t . Введем понятие пути m -го ранга. Простой $s-t$ -путь есть путь ранга 0. Простые контуры, имеющие общие вершины или дуги с путями нулевого ранга, образуют $s-t$ -пути первого ранга. Аналогично определяются пути произвольного ранга. С каждым таким путем связывается случайная величина — время его прохождения блуждающей точкой. Несмотря на бесконечность множества путей, для программ небольшого размера можно найти ожидаемое время блуждания и распределение этой случайной величины. Если же время исполнения оператора есть, в свою очередь, случайная величина, то мы имеем дело с полумарковской моделью. Процедура расчета среднего значения и дисперсии времени исполнения такой программы может быть сведена к пошаговой упрощающей замене фрагментов графа с последовательно и параллельно соединенными дугами (а также с петлями) новыми фрагментами с соответствующим рекуррентным пересчетом вероятности попадания на дугу q , среднего значения μ и дисперсии σ^2 времени для дуги (время исполнения операторов переносится на выходящие из соответствующей вершины дуги, что в общем все равно). На последнем шаге получается граф, состоящий из двух вершин и одной соединяющей их дуги, параметры μ и σ^2 которой дают искомые характеристики графа в целом. Этот же способ расчета можно применять и для марковской модели, если положить все $\sigma^2 = 0$, μ — равными фиксированным значениям времени исполнения операторов.

1.2. Операции над управляющими графами. Объединением управляющих графов $G = (X_1, U_1)$ с входом s_1 и выходом t_1 и $G_2 = (X_2, U_2)$ с входом s_2 и выходом t_2 называется управляющий граф $G = (X, U) = G_1 \cup G_2$

с входом s и выходом t , у которого $X = X_1 \cup X_2$, $U = U_1 \cup U_2$, а вход s и выход t получаются отождествлением входов s_1 и s_2 и выходов t_1 и t_2 соответственно.

Лемма 1. Для любых двух управляющих графов G_1 и G_2

$$G_1 \cup G_2 = G_2 \cup G_1.$$

Лемма 2. Для любого управляющего графа G

$$G \cup G = G.$$

Лемма 3. Для любых управляющих графов G_1 , G_2 и G_3

$$(G_1 \cup G_2) \cup G_3 = G_1 \cup (G_2 \cup G_3).$$

Пусть \bar{G} есть множество $s-t$ -путей в графе G .

Лемма 4. Для любых управляющих графов G_1 , G_2 , G_3 из $G_3 = G_1 \cup G_2$ следует $\bar{G}_3 \supseteq \bar{G}_1 \cup \bar{G}_2$.

Лемма 5. Для каждого управляющего графа G существует такое n и такие управляющие графы G_1, \dots, G_n , что

$$G = (\dots (G_1 \cup G_2) \cup \dots) \cup G_n.$$

Более того, для каждого $i_1, \dots, i_l \in \{1, \dots, n\}$ имеет место соотношение

$$\overline{(\dots (G_{i_1} \cup G_{i_2}) \cup \dots) \cup G_{i_l}} = \bar{G}_{i_1} \cup \dots \cup \bar{G}_{i_l} \quad (6.1)$$

(в частности, $\bar{G} = \bar{G}_1 \cup \dots \cup \bar{G}_n$).

Доказательство. Пусть управляющий граф $G = (X, U)$ имеет вход s и выходы q_{j_1}, \dots, q_{j_k} . Определим для каждого $l \in [j_1 : j_k]$ простой управляющий граф следующим образом: $G_l = (X_l, U_l, s, q_l)$, что означает наличие входа s и выхода q_l , и X_l есть множество всех тех вершин графа G , из которых достижима вершина q_l , а $U_l = \{(x, y) \in U \mid x, y \in X_l\}$. Очевидно, что $(\dots (G_1 \cup G_2) \cup \dots) \cup G_k = G$.

Докажем теперь соотношение (6.1). Пусть $i_1, \dots, i_l \in [1 : n]$. Достаточно показать, что имеет место включение

$$\overline{(\dots (G_{i_1} \cup G_{i_2}) \cup \dots) \cup G_{i_l}} \supseteq \bar{G}_{i_1} \cup \dots \cup \bar{G}_{i_l},$$

так как обратное включение следует из леммы 4. Но требуемое включение следует из того, что любой путь из входа s в какой-нибудь выход q_{i_j} принадлежит целиком графу G_{i_j} , а следовательно, и множеству \bar{G}_{i_j} .

Композицией $G_1 \circ G_2$ управляющих графов $G_1 = (X_1, U_1, s_1, t_1)$ и $G_2 = (X_2, U_2, s_2, t_2)$ называется управляющий граф $G = (X, U, s, t)$, у которого $X = X_1 \cup X_2$, $U = U_1 \cup U_2$, $s = s_1$, $t = t_2$, а выход t_1 отождествлен с входом s_1 .

Лемма 6. Для любых управляющих графов G_1, G_2, G_3 справедливы соотношения:

- а) $(G_1 \circ G_2) \circ G_3 = G_1 \circ (G_2 \circ G_3)$;
- б) $G_1 \circ (G_2 \cup G_3) = (G_1 \circ G_2) \cup (G_1 \circ G_3)$;
- в) $(G_1 \cup G_2) \circ G_3 = (G_1 \circ G_3) \cup (G_2 \circ G_3)$.

Лемма 7. Для любых управляющих графов G_1, G_2, G_3 из $G_3 = G_1 \circ G_2$ следует $\bar{G}_3 \equiv \bar{G}_1 \circ \bar{G}_2$.

Представим теперь введенные операции в виде синтезирующих графов (графов-операций) с формальными параметрами t_1, t_2, \dots , вместо которых подставляются управляющие графы (в простейшем случае это — элементарные графы e_0 и e_1 (рис. 6.1, в, г)). При подстановке вместо параметра t_i некоторого управляющего графа из него удаляются начальная и конечная

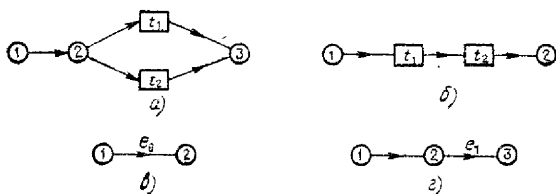


Рис. 6.1.

вершины; если подставляемый управляющий граф изоморфен e_0 , то t_i исчезает. Операции объединения (рис. 6.1, а) и композиции (рис. 6.1, б) не исчерпывают всех допустимых операций. Весьма продуктивными являются четыре типа итерации:

- итерация $I_1(t_1)$ типа while-do (рис. 6.2, а);
- итерация $I_2(t_1)$ типа repeat-until (рис. 6.2, б);
- итерация $I_3(t_1, t_2)$ типа for-while-do (рис. 6.2, в);
- итерация $I_4(t_1, \dots, t_n)$, $n \geq 1$ (рис. 6.2, г).

Графы, синтезируемые с помощью операций объединения, композиции и одной из четырех операций итерации $I_1 - I_4$, соответствуют одному из четырех классов структурированных управляющих графов.

1.3. Эквивалентные преобразования управляющих графов. Существует несколько преобразований управляющих графов, при которых исходный граф стано-

вится управляющим графом программы, функционально эквивалентной данной. В предлагаемых описаниях преобразований и описание, и иллюстрации будут касаться только той части графа, которая непосредственно затрагивается преобразованием. Особенно это касается вершины, обозначенной буквой N . Мы будем предполагать, что ни одна дуга не входит в нее и ни одна дуга не исходит из нее, кроме тех, которые изображены на рисунках.

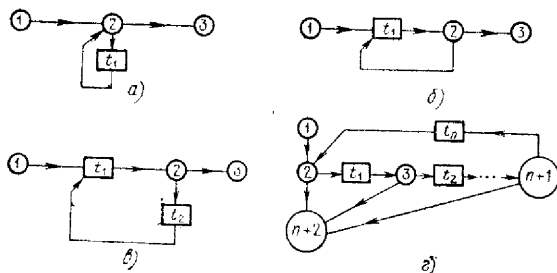


Рис. 6.2.

Преобразование Т1: расщепление вершины N . Пусть \bar{Q} — некоторое подмножество множества ΓN , $\bar{Q} \subseteq \Gamma N$, $\bar{Q} \neq \emptyset$. Удалим из графа все дуги, ведущие из N в \bar{Q} , добавим новую вершину N_1 , дугу (N, N_1) и дуги из N_1 в \bar{Q} . Заметим, что случай $N \in \Gamma N$ не исключается.

В примере на рис. 6.3 $\bar{Q} = \{N, D, E\}$.

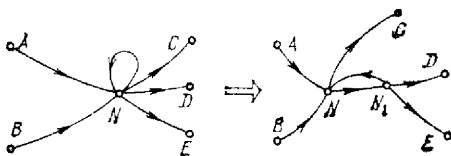


Рис. 6.3.

Преобразование Т2: расщепление вершины N . Пусть I — некоторое подмножество множества $\Gamma^{-1}N$, $I \subseteq \Gamma^{-1}N$, $I \neq \emptyset$. Удалим все дуги, ведущие из I в N , добавим новую вершину N_1 , дугу (N_1, N) и дуги, ведущие из всех вершин множества I в N_1 .

В примере на рис. 6.4 $I = \{A, B\}$.

Преобразование Т3: дублирование вершины N . Пусть $I \subset \Gamma^{-1}N$, $N \notin I$, $I \neq \emptyset$. Удалим все дуги,

ведущие из множества I в вершину N , добавим новую вершину N_1 , дуги из N_1 во все вершины из множества ΓN , исключая N , если есть петля (N, N) , и дуги из каждой вершины из множества I в N_1 . Если есть петля (N, N) , то добавим петлю (N_1, N_1) .

На рис. 6.5 $I = \{A, B\}$.

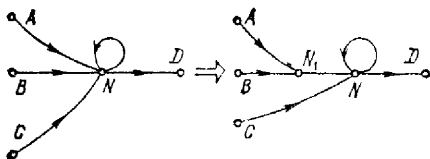


Рис. 6.4.

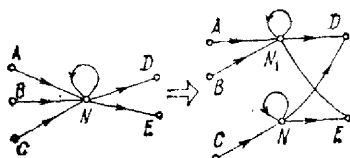


Рис. 6.5.

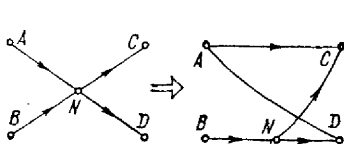


Рис. 6.6.

Преобразование T4: удаление дуги (A, N) . Это преобразование можно провести только в том случае, когда нет петли (N, N) . Удалим дугу (A, N) и добавим дуги, ведущие из вершины A во все вершины множества ΓN . Если $\Gamma^{-1}N = \{A\}$, то удаляем также вершину N и все дуги, исходящие из N .

Схема данного преобразования дана на рис. 6.6.

Преобразование T5: удаление вершины N . Это преобразование может быть проведено только тогда, когда нет петли (N, N) ; оно представляет собой последовательность преобразований T4, примененных в произвольном порядке к дугам, идущим в вершину

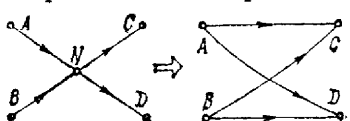


Рис. 6.7.

N . Состоит оно в удалении всех дуг, заходящих в вершину N и исходящих из N , удалении вершины N и добавлении дуг, ведущих из каждой вершины множества $\Gamma^{-1}N$

в каждую вершину множества N .

Схема данного преобразования дана на рис. 6.7.

Из данных преобразований наиболее сильные изменения в граф вносят преобразования T4 и T5. Остановимся подробнее на их свойствах.

Поскольку вершина может быть удалена только в том случае, когда в ней нет петли, то удаление вершин следует проводить так, чтобы получить петлевой граф (см. п. 5.9 главы 2). Возникают два вопроса: каким необходимым и достаточным условиям должен удовлетворять граф, чтобы его можно было бы преобразовать путем последовательного удаления вершин в петлевой граф, и в каком порядке следует удалять вершины?

Теорема 1. Если орграф является KLC-графом, то его нельзя преобразовать в петлевой граф.

Утверждение теоремы очевидно, так как по определению k -контурно-связного графа последний обязательно содержит контур, отличный от петли.

Будем говорить, что вершина N может быть безопасно удалена из графа G , давая в результате граф G' , тогда и только тогда, когда она может быть удалена с помощью преобразования Т5 и в графе G найдется вершина A , отличная от N и обладающая тем свойством, что все простые контуры в графе G , содержащие N , содержат и A .

Укажем некоторые частные случаи, когда вершина N может быть безопасно удалена:

- если не существует контура, содержащего вершину N ;

- если только один простой контур содержит вершину N ;

- если из вершины N исходит только одна дуга (отличная от петли);

- если в вершину N заходит единственная дуга.

Пусть теперь в произвольном графе осуществляются в произвольном (!) порядке безопасные удаления вершин, пока не получится граф, в котором данную операцию выполнить нельзя.

Теорема 2. Если граф G не является k -контурно-связным и граф G' получается из G безопасным удалением вершины N , то граф G' также не является k -контурно-связным ни для какого k .

Теорема 3. Если граф G не является k -контурно-связным ни для какого k и если ни одна вершина G не может быть безопасно удалена, то граф G является петлевым.

Теорема 4. Если из графа G удалена вершина N , и удаление не является безопасным, то получившийся граф G' является k -контурно-связным для некоторого k .

Доказательства теорем мы опускаем из-за их громоздкости.

Теорема 4 показывает, что безопасные удаления представляют собой самый широкий класс удалений, которые можно использовать в описанном процессе.

Введенное в главе 2 понятие контурного ранга дает возможность по-новому определить безопасное удаление вершины. Чтобы отличить этот вариант безопасного удаления от введенного, будем говорить о *C-удалении*; вершину же, которую можно безопасно удалить *C-удалением*, будем называть *безопасной*.

Удаление вершины N из графа G называется (безопасным) *C-удалением*, если существует минимальный разрез в графе G , не содержащий вершины N .

Теорема 5. Если граф G' получается из графа G *C-удалением* вершины, то $k_G = k_{G'}$, где k_G — контурный ранг G .

Теорема 6. Если граф G' получается из графа G удалением небезопасной вершины, то $k_{G'} > k_G$.

Утверждения лемм очевидны.

Лемма 8. Если минимальное сечение в графе G содержит все вершины графа, то тогда каждая вершина имеет петлю.

Лемма 9. Если в графе невозможно ни одно *C-удаление* вершины, то граф имеет петлю в каждой вершине.

Теорема 7. Если G — сильно связный граф и G имеет петлю в каждой вершине, то граф G может быть преобразован в полный орграф с n вершинами с помощью преобразования T_4 удаления дуг.

Доказательство. Пусть A_{i_1} и A_{i_2} — две вершины графа $G = (X, U)$ такие, что $(A_{i_1}, A_{i_2}) \in U$. Так как граф G сильно связан, то существует путь $[A_{i_1}, \dots, A_{i_r}]$ из A_{i_1} в A_{i_r} . Если теперь удалить дугу $(A_{i_j}, A_{i_{j+1}})$ из графа G , то тогда, по определению преобразования T_4 , добавятся дуги $(A_{i_{j-1}}, A_{i_{j+1}})$ и $(A_{i_j}, A_{i_{j+2}})$. Продолжая процесс, мы получим в итоге, что в дополнение к пути $[A_{i_1}, \dots, A_{i_r}]$ появится дуга (A_{i_1}, A_{i_r}) , откуда и следует утверждение теоремы.

Дуга u графа G называется *безопасной*, если u есть дуга простого контура в графе G и конец дуги u есть безопасная вершина. Удаление дуги есть *безопасное удаление*, если удаляемая дуга безопасна.

Теорема 8. Если граф G' получается из графа G безопасным удалением дуги, то $k_{G'} = k_G$.

Следствие. Если граф G' получается из графа G удалением дуги, не содержащейся ни в одном контуре в G , то $k_{G'} = k_G$.

Теорема 9. Если граф G' получается из графа G удалением дуги (A, B) , которая принадлежит простому контуру в G , но ее конец B не является безопасной вершиной, то $k_{G'} > k_G$.

1.4. Связь контурной связности с контурным рангом. Легко увидеть, что если вершина допускает безопасное удаление, то она допускает и C -удаление, хотя обратное не обязательно верно. Предположим теперь, что G — сильно связный граф с $k_G \geq 2$. Пусть граф G' получается из графа G применением безопасных удалений вершин, пока это возможно. Тогда по теореме 8 $k_{G'} = k_G \geq 2$. Следовательно, граф G' также сильно связан. Предположим теперь, что граф G не контурно-связный граф. Тогда по теореме 3 граф G' является петлевым. Но граф G' имеет по крайней мере две вершины и сильно связан, поэтому он не может быть петлевым, следовательно, он должен быть контурно-связным. Обратно, если G контурно-связен, то существуют вершины A, B в G и контуры \bar{C}_A, \bar{C}_B и \bar{C} , которые удовлетворяют определению контурно-связного графа. В частности, в графе не имеется ни одной общей для \bar{C}_A, \bar{C}_B и \bar{C} вершины, что влечет $k_G \geq 2$. Таким образом, если граф G сильно связан, то он контурно-связен тогда и только тогда, когда $k_G \geq 2$.

Предположим, что G — произвольный орграф. Рассмотрим его бикомпоненты. Если каждая из них имеет контурный ранг 1, то последовательным применением безопасных удалений вершин граф редуцируется к петлевому графу; в противном случае он будет редуцирован к графу, являющемуся LC -графом, и, следовательно, граф G не преобразуется в петлевой граф.

Пусть теперь граф G' есть петлевой граф. Ясно, что контурный ранг каждой его бикомпоненты равен 1, и отсюда следует, что то же верно для каждой бикомпоненты графа G . Таким образом, граф G преобразуется в петлевой граф с помощью безопасных удалений тогда и только тогда, когда ранг каждой бикомпоненты графа G равен 1. По сказанному выше это верно только тогда, когда каждая бикомпонента

графа G не является контурно-связной, что справедливо, если сам граф не является контурно-связным. Следовательно, мы доказали, что граф G преобразуется к петлевому графу, если он не LC -граф.

1.5. Цикломатическая мера сложности программ. Определим цикломатическое число орграфа как цикломатическое число неориентированного графа, получающегося заменой всех дуг исходного орграфа ребрами.

Теорема 10. *В сильно связном орграфе цикломатическое число равно числу линейно независимых контуров.*

Применяется эта теорема так. Пусть программа представлена своим управляющим графом с входом s и выходом t . Если соединить t с s дугой, то получится сильно связный граф, и, следовательно, по теореме 10 можно вычислить число линейно независимых контуров. Это дает возможность выразить любой $s-t$ -путь как линейную комбинацию независимых контуров.

Пример. Пусть управляющий граф некоторой программы имеет вид, показанный на рис. 6.8 (штриховая линия — добавленная дуга). Цикломатическое число этого графа равно 5, т. е. в графе существует пять независимых контуров. В качестве их можно выбрать, например, контуры $[s, v_1, v_4, t, s]$, $[v_1, v_2, v_3]$, $[s, v_1, v_2, s]$, $[s, v_2, t, s]$, $[s, v_3, v_2, t, s]$. Тогда путь $[s, v_1, v_4, s, v_1, v_4, v_1, v_4, v_1, v_4, t]$ выражается в виде линейной комбинации $[s, v_1, v_4, s] + 2[v_1, v_2, v_3] + [s, v_1, v_4, t, s]$.

Ясно, что основная идея оценки сложности программы через цикломатическое число состоит в измерении сложности программы путем вычисления числа линейно независимых путей $v(G)$ и в контроле «размера» программы наложением ограничения сверху на $v(G)$ (вместо использования чисто физического размера).

Цикломатической сложностью программы называется величина $v(G) = \lambda(G) + 2$, где $\lambda(G)$ есть цикломатическое число соответствующего управляющего графа.

Укажем на некоторые важные свойства цикломатической сложности:

1) $v(G) \geq 1$.

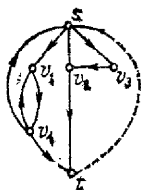


Рис. 6.8.

2) $v(G)$ равно максимальному числу линейно независимых путей в графе G .

3) Изменение числа функциональных операторов в программе не изменяет $v(G)$.

4) Граф G имеет единственный $s-t$ -путь тогда и только тогда, когда $v(G) = 1$.

5) Добавление одной дуги увеличивает $v(G)$ на единицу.

6) $v(G)$ зависит только от логической структуры управляющего графа программы.

В общем случае существует корреляция между относительными цикломатическими сложностями программ и интуитивным ощущением их сложности. Однако когда «неструктурированная» программа сравнивается с некоторой хорошо структурированной версией программы, функционально эквивалентной исходной, то оказывается, что цикломатическая сложность неструктурированной версии меньше, чем структурированной.

Цикломатическая сложность есть только одна компонента в измерении общей сложности программы. Хорошо структурированная программа более ясно выражена, ее легче читать, она более понятна, более легка для проверки и отладки. С другой стороны, ясно, что программа с цикломатическим числом более 10 создает впечатление ненадежной. Совершенно ясно, что цикломатическую сложность опасно использовать для прямого сравнения программ, поэтому имеет смысл использовать ее как базу для получения более общих результатов.

1.6. *Тестирование* — это проверка правильности программы с помощью множества контрольных примеров — тестов.

Тестом, проверяющим программу, называется тройка $\tau = \langle \hat{x}, \mu(G), \hat{y} \rangle$, где $\hat{x} \subset \hat{X}$ — подмножество значений входных данных; $\mu(G)$ — $s-t$ -путь в управляющем графе G , соответствующий входным данным \hat{x} , $\hat{y} \subset \hat{Y}$ — подмножество значений выходных данных при прохождении программой пути \hat{y} .

Из определения вытекает, что для полного тестирования программы желательно проверить все $s-t$ -пути. Но так как это невозможно даже для небольших программ, то возможны различные стратегии тестирования.

В простейшем случае задача тестирования сводится к проверке предписанного числа операторов в программе. Это требует построения пути в управляющем графе, который проходит через предписанное множество вершин, если таковой существует. Естественно искать при этом не просто путь, а кратчайший путь. В общем случае следует предполагать, что каждой вершине сопоставлен вес $w > 0$, который можно интерпретировать как сложность оператора (или отдельного блока) программы, изображаемого данной вершиной. Можно ожидать, что отыскание тестирующих данных для тестирующего пути будет проще, если в нем будет содержаться как можно меньше вершин из множества L вершин, не подлежащих проверке. Это соответствует задаче отыскания $s - t$ -пути с наибольшим весом при условии, что вершинам из множества L приписан небольшой отрицательный вес. Решение этих задач было рассмотрено в главе 3. Более общий случай требует построения тестирующих путей, покрывающих или все требуемые пути (заданные отрезки путей в управляющем графе), или все дуги, или все вершины управляющего графа. Эти задачи сводятся к построению различного вида покрытий графа путями.

Рассмотрим метод тестирования, основанный на цикломатической сложности и сводящийся к построению множества $s - t$ -путей, каждый из которых содержит дугу, не входящую ни в какой другой путь. Предположим, что программа \mathcal{P} уже написана, вычислена ее цикломатическая сложность v и определено число путей \bar{n} для тестирования (фактическая сложность). Если $\bar{n} > v$, то должно выполняться одно из следующих условий:

а) должно быть протестировано большее число путей;

б) управляющий граф программы может быть редуцирован к графу со сложностью $v - \bar{n}$ (т. е. может быть удалено $v - \bar{n}$ логических операторов);

в) части программы могут быть редуцированы в линейный код (т. е. сложность увеличивается для сохранения объема занятой памяти).

Приведем несколько простых примеров, поясняющих сказанное. Предположим, что имеется управляющий граф (рис. 6.9) с цикломатической сложностью $v(G)$, равной 3. Предположим, что $\bar{n} = 2$ и в качестве

тестирующих путей взяты пути $[s, v_1, v_3, v_5, t]$ и $[s, v_2, v_3, v_4, t]$. При этом не проверяются еще два возможных пути, имеем $\bar{n} < v$, т. е. реализуется случай б), и граф G может быть редуцирован к графу G_1 удалением оператора (рис. 6.10). Заметим, что для G_1 имеет место равенство $v = \bar{n}$ и сложность графа G_1

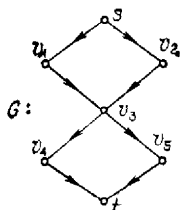


Рис. 6.9.

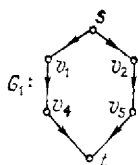


Рис. 6.10.

меньше сложности G . На практике этот подход полезен тогда, когда программистам требуется документировать свои управляющие графы и показывать точно различные тестирующие пути. Часто случается, что при сравнении числа тестируемых путей с цикломатической сложностью выявляются новые тестирующие пути, которые при другом подходе были бы не замечены. Хотя есть смысл считать, что цикломатическая сложность v равна минимальному числу путей, подлежащих тестированию, на практике используются методы тестирования как с большим, так и с меньшим числом путей.

В качестве примера укажем на способ тестирования, при котором число тестирующих путей не превосходит цикломатической сложности, но который гарантирует:

а) прохождение каждой вершины по крайней мере один раз;

б) прохождение каждой вершины с двумя исходящими дугами (логический оператор) по крайней мере один раз по каждой исходящей дуге, хотя и не во всех возможных комбинациях.

Используемые на практике подходы часто учитывают информационные связи в программе и излагаются в специальной литературе.

1.7. Задача о контрольных точках. Задача возникает при отладке больших программ и состоит в нахождении наименьшего числа точек программы, в которые

нужно поместить операторы выдачи контрольных распечаток для обнаружения ошибок в функционировании циклических частей программы. В теории графов эта задача соответствует задаче отыскания наименьшего по мощности множества дуг, удаление которых разрывает все контуры и тем самым превращает оргграф в бесконтурный. Нахождение точного, решения представляет собой *NP*-полную проблему, и для ее решения неизвестен эффективный алгоритм. Рассмотрим приближенное решение, при котором отыскивается множество дуг, принадлежащих как можно большему числу контуров.

Вход: Множество простых контуров управляющего графа, заданных перечислением дуг, входящих в контуры.

Выход: Множество K дуг, разрывающих все контуры.

начало

1. построить матрицу $H = \|h_{ij}\|$; контуры-дуги которой

$$h_{ij} = \begin{cases} 1, & \text{если дуга } u_j \text{ принадлежит контуру } C_i \\ 0 & \text{в противном случае;} \end{cases}$$

2. включить в множество K те дуги, которым в матрице H соответствуют строки, содержащие только одну единицу;

примечание тем самым в множество K включаются все петли графа **конец примечания**

3. удалить из H строки и столбцы, соответствующие этим петлям;

4. пока H не пусто цикл

5. исключить из H все мажорируемые столбцы, т. е. те, для которых найдется хотя бы один столбец, имеющий 1 в тех же строках, что и рассматриваемые;

6. включить в множество K дугу u_r , соответствующую столбцу с наибольшим числом единиц;

7. вычеркнуть из H строки, соответствующие единицам в r -м столбце, и сам r -й столбец;

конец

Алгоритм основан на принципе поиска локального оптимума, его трудоемкость составляет $O(ct)$, где c — число контуров, а t — число дуг.

§ 2. Применение методов теории графов к организации вычислительного процесса

2.1. Размещение блоков программы в оперативной памяти ЭВМ: сведение к задаче коммивояжера. Пусть X_1, \dots, X_n — отдельные блоки программы, и пусть p_{ij} — вероятность исполнения блока X_j после X_i . Если X_1 — первый блок, а X_n — последний, то положим $p_{n1} = 1$ и $p_{nj} = 0$. Если марковский процесс, описываемый матрицей переходов $p = \|p_{ij}\|$, является эргодическим, т. е. для него существует предельное распределение и оно не зависит от начального состояния процесса, то частоты исполнения отдельных блоков N_1, \dots, N_n даются решением $N = (N_1, \dots, N_n)$ матричного уравнения $NP = N$, нормализованного так, что сумма частот исполнения равна 1. Пусть величины N_i найдены. Тогда, если τ_i — время исполнения блока X_i , то ожидаемое время исполнения всей программы равно $\sum_{i=1}^n \tau_i N_i$. Если p_{ij} и N_i известны, то можно рассмотреть задачу оптимизации: минимизировать ожидаемое число передач управления, производимых при выполнении программы. Если блок X_j следует в памяти за блоком X_i , то передача управления пуста всякий раз, когда после блока X_i будет выполняться блок X_k , отличный от X_j . Ожидаемое число таких передач равно $c_{ij} = N_i(1 - p_{ij})$.

Пусть $x_{ij} = 1$, если блок X_j непосредственно следует за блоком X_i в памяти, и $x_{ij} = 0$ в противном случае. Пусть X_{n+1} — блок, состоящий из кодов начальных данных и кодов, которые не выполняются в виде команд. Тогда для всех i имеем $c_{i, n+1} = c_{n+1, i} = 0$. Полагаем $x_{i, n+1} = 1$, если X_i — последний блок в памяти, и $x_{n+1, i} = 1$, если X_i — первый блок в памяти, в остальных случаях полагаем $x_{i, n+1} = x_{n+1, i} = 0$. Тогда рассматриваемая задача сводится к минимизации ожидае-

мого числа выполнения передач управления $\sum_{i=1}^{n+1} \sum_{j=1}^{n+1} c_{ij} x_{ij}$ при условии, что $\|x_{ij}\|$ есть матрица циклической перестановки, а это и есть задача коммивояжера. Для решения задачи коммивояжера не существует эффективных алгоритмов, она является NP -полной проблемой. Она также является эталонной при исследовании трудоемкости в том смысле, что многие задачи легко сводятся к ней (за полиномиальное время).

2.2. Рассмотрим ситуацию, когда на входе ЭВМ имеется некоторое множество задач, имеющих общие страницы в памяти. Требуется определить такую последовательность пропуска этих задач, чтобы минимизировать пересылки страниц памяти. Для решения построим граф, вершинами которого будут служить задачи, а две вершины будут соединены ребром тогда и только тогда, когда они соответствуют задачам, имеющим общие страницы памяти. Каждому ребру сопоставляется вес, зависящий от числа общих страниц. Тогда *проблема активизации задач* сводится к отысканию оптимального каркаса графа активизации. Особенность этой проблемы состоит в том, что чаще всего стоимость каркаса есть нелинейная функция от весов ребер.

2.3. Минимизация памяти при вычислении арифметических выражений. Арифметические выражения могут быть изображены корневыми ордеревьями, в каждую вершину которых заходит не более двух дуг. Начальные данные соответствуют висячим вершинам дерева, а промежуточные результаты — внутренним вершинам. Каждая внутренняя вершина изображает бинарную операцию над аргументами, соответствующими ее предшественникам. Порядок, в котором нужно брать аргументы, в простейших моделях не учитывается, в более сложных моделях он может играть важную роль.

Пример. Процесс вычисления арифметического выражения $((a + b) - c * d) / (e * (f - g))$ изображается деревом на рис. 6.11.

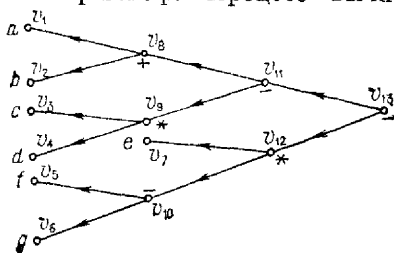


Рис. 6.11.

Выбор возможный порядок операций — значит топологически упорядочить дерево, т. е. расположить его вершины в целочисленных точках прямой в

такой последовательности, что вершина v предшествует вершине \bar{v} , если существует дуга из \bar{v} в v . Это условие эквивалентно требованию, чтобы каждый промежуточный результат был вычислен раньше, чем использован.

Пример топологического упорядочения дерева, изображенного на рис. 6.11, показан на рис. 6.12.

Предположим, что мы сделали «моментальный снимок» вычисления в момент выполнения операции, изображаемой вершиной v . Легко увидеть, что величины, не использующиеся в качестве аргументов в операции v , должны храниться в памяти; на рис. 6.12

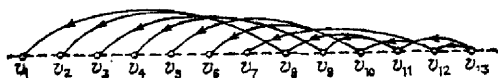


Рис. 6.12.

этим величинам соответствуют дуги, проходящие над вершиной v . Если число этих дуг равно $w(v)$, то мы должны иметь не менее $w + 1$ ячеек памяти в нашем распоряжении в момент выполнения операции v : w ячеек памяти для хранения промежуточных результатов (тех, которые уже находятся в памяти) и одну для результата операции v . Таким образом, возникает оптимизационная задача: минимизировать количество ячеек памяти при организации вычисления арифметического выражения. Эта задача сводится к построению укладки ордера с минимальной шириной.

2.4. Оптимальная укладка дерева модулей. В модульном программировании задача есть определяющий объект, которому подчиняются остальные, логически связанные между собой объекты, называемые *разделами*. Разделы являются результатом сборки отдельных оттранслированных программ. Таким образом, задача представляется в виде нерархической структуры, имеющей форму дерева. Вершины дерева суть модули — разделы и программы, а дуги определяют их взаимосвязь. С помощью специальной управляющей программы производится загрузка разделов в оперативную память. Оптимальное размещение информации — это такой порядок следования модулей в задаче, при котором наиболее экономно расходуется оперативная память ЭВМ.

Пусть T есть дерево, изображающее нерархическую структуру задачи, и пусть каждой дуге u или, что то же самое, вершине v , являющейся концом дуги u , приписан вес $R_u(R_v) \geq 0$. Кроме того, будем считать, что каждой вершине v дерева приписана дополнительно некоторая числовая характеристика $P_v \geq 0$, называемая

мая дополнительным весом вершины. С содержательной точки зрения вес R_v может обозначать размер памяти, требуемой для хранения результата операции, выполняемой в вершине v , а дополнительный вес P_v — размер памяти для хранения команд или размер дополнительной части рабочего поля. Проблема оптимального использования оперативной памяти при решении иерархически организованной задачи сводится к нахождению оптимальной по ширине укладки взвешенного ордерера. Заметим, что рассмотренная в главе 3 оптимальная по длине укладка соответствует такому порядку выполнения операций, при котором минимизируется суммарное время хранения промежуточных результатов. Возможен вариант этой проблемы, при котором минимизируется наибольшее время хранения промежуточного результата. Оптимальная по числу соседней связности укладка соответствует порядку выполнения программы, при котором увеличивается интенсивность использования регистровой памяти, что ведет к уменьшению потерь времени, затрачиваемого в процессе выполнения программы на обращения к оперативной памяти.

2.5. Генерация оптимального кода для арифметических выражений. Пусть имеется машина с неограниченной памятью и N общими регистрами, и пусть допустимыми являются команды следующих типов:

1. Пересылка: память \rightarrow регистр.
2. Пересылка: регистр \rightarrow память.
3. Операция: (регистр, память) \rightarrow регистр.
4. Операция: (регистр, регистр) \rightarrow регистр.

Заметим, что команда типа

Операция: (память, регистр) \rightarrow регистр

недопустима. Это ограничение согласуется с системой команд многих ЭВМ.

Будем понимать под программой последовательность операторов, вычисляющих данное выражение для начальных данных, хранящихся в выделенных ячейках памяти. Под стоимостью вычислений удобно понимать количество программных шагов, требуемых для полного завершения процесса вычисления.

Чтобы определить минимальное число требуемых регистров, а также оптимальную последовательность операций, будем использовать разметку вершин, считая вначале, что все операции некоммутативные. Расстановку меток будем производить следующим образом:

1. Если вершина v висячая и является левым потомком некоторой другой вершины, то полагаем $l(v) = 1$; если она является правым потомком, то полагаем $l(v) = 0$.

З а м е ч а н и е. Мы предполагаем, что дерево изображено на плоскости так, что относительно любой пары дуг, исходящих из вершины дерева, можно сказать, какая дуга левая, а какая правая.

2. Если вершина v имеет потомков с метками l_1 и l_2 , то при $l_1 \neq l_2$ полагаем $l(v) = \max\{l_1, l_2\}$, а при $l_1 = l_2 = l$ полагаем $l(v) = l + 1$.

При описании алгоритма будем говорить о значении вершины, подразумевая под этим величину (данные, промежуточный результат), которую она изображает, и о вычислении вершины, подразумевая под этим исполнение операции, приданной этой вершине.

В х о д: Ордерное арифметическое выражение, заданное списками смежности, с размеченными вершинами; регистры b_1, \dots, b_N .

В ы х о д: Программа вычисления выражения с оптимальным использованием регистров.

начало

1. процедура $T(v)$;

2. **начало**

3. **если** $l(v) = 1$ **то**

4. **если** v — висячая вершина **то**

5. заслать значение вершины v в доступный регистр b_m с наименьшим номером;

примечание в этом случае вершина v является левым потомком своего предка **конец примечания**

6. **иначе** $T(\alpha(v))$;

7. **иначе если** $\min\{l(\alpha(v)), l(\beta(v))\} \geq N$ **то**

примечание мы используем обозначение $\beta(v)$ для идентификации правого потомка и обозначение $\alpha(v)$ для идентификации левого потомка **конец примечания**

8. $T(\beta(v))$;

9. **иначе если** $l(\alpha(v)) \neq l(\beta(v))$ **то**

10. **начало**

11. $w :=$ потомок вершины v с большей меткой;

12. $T(w)$;

13. **конец**

14. **иначе** $T(\beta(v))$;

15. **если** $l(v) = 1 \wedge v$ не есть висячая вершина **то**

16. **начало**

17. вычислить вершину v , взяв значение левого потомка из регистра b_m , а значение правого потомка из памяти;

18. заслать значение вершины v в регистр b_m ;

19. **конец**

20. **иначе начало**

21. $T(\alpha(v))$;

22. выполнить операцию, соответствующую вершине v , беря значение левого потомка из регистра b_m , а значение правого потомка из регистра b_{m+1} ;

примечание на этом шаге доступными являются регистры b_m и b_{m+1} **конец** примечания

23. **если** $l(v) \geq N \wedge v$ — правый потомок своего предка **то**

24. **начало**

25. значение вершины v заслать в память;

26. освободить регистры b_m и b_{m+1} ;

27. **конец**

28. **конец**

29. **конец** процедуры $T(v)$;

30. $v :=$ корень дерева T ;

конец

Алгоритм состоит из двух частей: первая дает оптимальную укладку, вторая — организует вычисления с регистрами.

Пример. Рассмотрим выражение

$$((ab - c)/(d + e))/(g + i)/(j + k) * l - m),$$

дерево для которого показано на рис. 6.13. Числа, расставленные возле вершин, суть метки $l(v)$.

Для $N = 2$ алгоритм генерирует код:

1. $j \rightarrow$ РЕГИСТР b_1

2. РЕГИСТР $b_1 + k \rightarrow$ РЕГИСТР b_1

3. РЕГИСТР $b_1 * l \rightarrow$ РЕГИСТР b_1

4. РЕГИСТР $b_1 - m \rightarrow$ РЕГИСТР b_1

5. $g \rightarrow$ РЕГИСТР b_2

6. РЕГИСТР $b_2 + i \rightarrow$ РЕГИСТР b_2

7. РЕГИСТР $b_1/\text{РЕГИСТР } b_2 \rightarrow$ (память)

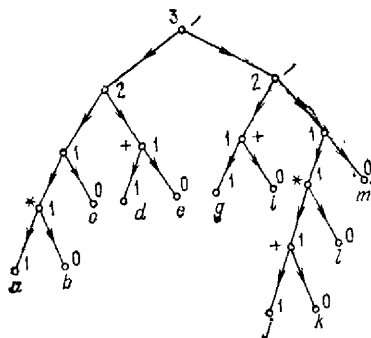


Рис. 6.13.

8. $d \rightarrow \text{РЕГИСТР } b_1$
9. $\text{РЕГИСТР } b_1 + e \rightarrow \text{РЕГИСТР } b_1$
10. $a \rightarrow \text{РЕГИСТР } b_2$
11. $\text{РЕГИСТР } b_2 * b \rightarrow \text{РЕГИСТР } b_2$
12. $\text{РЕГИСТР } b_2 - c \rightarrow \text{РЕГИСТР } b_2$
13. $\text{РЕГИСТР } b_1 / \text{РЕГИСТР } b_2 \rightarrow \text{РЕГИСТР } b_1$
14. $\text{РЕГИСТР } b_1 / (\text{память}) \rightarrow \text{РЕГИСТР } b_1$

Покажем, что описанный алгоритм генерирует программу наименьшей длины, если все операции считаются некоммутативными.

Лемма 10. Пусть алгоритм начинает работу с N доступными регистрами, и пусть r есть число регистров, которые доступны в момент, когда рассматривается вершина v . Тогда $r = N$, если $l(v) > N$, и $N \geq r \geq l(v)$, если $l(v) \leq N$.

Лемма доказывается индукцией по числу вершин, рассматривавшихся алгоритмом.

Лемма 11. Алгоритм строит программу без пересылок результатов из регистра в память (т. е. без обращения к памяти), если доступно столько регистров, какова метка корня.

Лемма 12. Для дерева с корнем, помеченным меткой 2, метка корня есть нижняя оценка минимального числа регистров, необходимых для построения программы без обращения к памяти.

Лемма 13. Метка вершины есть нижняя граница для наименьшего числа регистров, требуемых для вычисления вершины без обращения к памяти.

Из лемм 11 и 13 непосредственно вытекает

Теорема 11. При условии отсутствия нетривиальных отношений между операторами и данными алгоритм использует минимальное число регистров для построения программы без обращения к памяти.

Для числа регистров N будем называть вершину v старшей, если оба ее потомка имеют метки не меньше N , и младшей, если она висятая и является левым потомком.

Лемма 14. Минимальное число обращений к памяти при вычислении арифметического выражения без нетривиальных отношений между операторами и данными не меньше числа старших вершин в дереве этого выражения.

Доказательство леммы проводится индукцией по числу вершин.

Лемма 15. Число обращений к памяти при построении программы равно числу старших вершин в дереве.

Утверждение леммы следует из того, что обращение к памяти происходит только при вычислении старшей вершины.

Лемма 16. Число засылок в регистры при построении программы для данного дерева равно по крайней мере числу младших вершин в дереве.

Доказательство леммы проводится индукцией по числу младших вершин.

Лемма 17. Алгоритм строит программу, используя столько засылок в регистр, сколько в дереве имеется младших вершин.

Теорема 12. Для выражений с некоммутативными операциями и без нетривиальных отношений между операторами и данными алгоритм генерирует программу с минимальным числом шагов.

В самом деле, минимальность обращений к памяти следует из лемм 14 и 15. Минимальность числа засылок в регистры следует из лемм 16 и 17. Каждая специфицированная операция выполняется один раз; следовательно, число программных шагов минимально.

2.6. Модифицированный алгоритм генерации оптимального кода. Если допустить коммутативность операций, то исходное дерево можно модифицировать, добиваясь уменьшения числа младших вершин и величины метки корня. Алгоритм изменяется следующим образом:

если у вершины v с меткой $l(v) > 1$ левый потомок есть висячая вершина и соответствующая операция коммутативна, то поменяем потомков местами так, чтобы висячая вершина стала правым потомком, причем ограничение $l(v) > 1$ сохраняется, если обе вершины висячие.

Алгоритм, в который внесены соответствующие изменения, будем называть коммутативным алгоритмом или К-алгоритмом.

Теорема 13. Коммутативный алгоритм порождает оптимальную последовательность операций при вычислении арифметического выражения, если только в выражение входят коммутативные и некоммутативные операции и не существует других нетривиальных отношений между операторами и данными.

Дальнейшее уменьшение числа требуемых регистров и времени вычислений может быть достигнуто, если наряду со свойством коммутативности учитывать также и свойство ассоциативности.

Для простоты будем рассматривать только те операции, которые одновременно коммутативны и ассоциативны (АК-операции).

Назовем *фрагментом* множество вершин дерева, такое, что:

1) каждая вершина в множестве сопоставлена некоторой операции;

2) ребра, соединяющие вершины множества, образуют дерево;

3) если вершине сопоставлена операция, не являющаяся АК-операцией, то эта вершина образует отдельное множество.

Корень фрагмента — это вершина в фрагменте, которая предшествует всем другим вершинам фрагмента. *Предок* фрагмента — это предок корня фрагмента. *Потомки* фрагмента — это потомки его висячих вершин. *Поддеревья-потомки* фрагмента — это поддеревья, которым предшествуют висячие вершины фрагмента. Наконец, *максимальный фрагмент* — это фрагмент, никакое собственное подмножество которого фрагментом не является. Охарактеризуем максимальные фрагменты данного бинарного дерева. Пусть максимальный фрагмент S имеет m потомков $\delta_1, \dots, \delta_m$, и пусть корень фрагмента S является потомком максимального фрагмента A . Очевидно, что $\delta_1, \dots, \delta_m$ — корни фрагментов; пусть это будут фрагменты D_1, \dots, D_m . Стынем каждый максимальный фрагмент в отдельную вершину. Получившееся дерево назовем *ассоциативным деревом*.

Приведенные определения имеют следующую основу. Пусть T — бинарное дерево и A — соответствующее ему ассоциативное дерево. Пусть T' образуется из T с помощью одного ассоциативного или коммутативного преобразования. Тогда ассоциативное дерево A' , соответствующее дереву T' , может быть получено из дерева A перестановкой потомков одной вершины. Такая вершина соответствует максимальному фрагменту в дереве T . Обратно, любая перестановка потомков вершины в дереве A может быть отражена в виде применения последовательности АК-преобразований в некотором фрагменте дерева T .

Пусть F — семейство деревьев, порождаемых из дерева T АК-преобразованиями. Благодаря такому подходу задача нахождения члена семейства F , генерирующего лучшую программу с помощью алгоритма п. 2.5, разбивается на две части. Вначале, используя перестановки потомков ассоциативных вершин, мы находим «лучшее» упорядочение для потомков. Потом из полученного ассоциативного дерева строим бинарное дерево, которое порождает минимальную программу.

Правила расстановки меток в ассоциативном дереве:

1. Если v — висячая вершина и самый левый потомок ее предка, то $l(v) = 1$. Для всех других висячих вершин $l(v) = 0$.

2. Если вершина v имеет m потомков с метками l_1, \dots, l_m , причем $l_1 \geq l_2 \geq \dots \geq l_m$, то $l(v) = l_1$, если $l_1 > l_2$, и $l(v) = l_1 + 1$, если $l_1 = l_2$.

Схема алгоритма с учетом ассоциативности:

1. Свернуть дерево в ассоциативное дерево.

2. Расставить метки согласно правилам для ассоциативных деревьев и перейти к выполнению шагов 3 и 4 снизу вверх.

3. Для каждой ассоциативной вершины v упорядочить ее потомков $\delta_0, \delta_1, \dots, \delta_m$ так, что $l(\delta_0) \geq l(\delta_1) \geq \dots \geq l(\delta_m)$. Если возможно, то сделать δ_0 невисячей вершиной. Заменить ассоциативную вершину v вершинами $v_i, i \in [1:m]$, так что v_{i+1} становится потомком вершины $v_i, i \in [1:m]$. Тогда левый потомок вершины v_{i+1} есть v_i , а правый потомок — вершина $\delta_i, i \in [1:m]$. Левым и правым потомками вершины v_i являются δ_0 и δ_i соответственно. Предок вершины v есть теперь предок вершины v_m .

4. Если вершина v неассоциативная, то действовать, как предписано в К-алгоритме.

Лемма 18. Пусть N — число доступных регистров. Тогда для данного бинарного дерева, содержащего фрагмент с t поддеревьями-потомками, r из которых имеют корни с метками, большими или равными N , фрагмент содержит по крайней мере $r - 1$ старших вершин.

Лемма доказывается индукцией по длине максимального пути P поддерева, доминируемого фрагментом, т. е. дерева, образованного рассматриваемым фрагментом и его поддеревьями-потомками.

Лемма 19. Алгоритм с учетом ассоциативности порождает дерево T , вычисляющее тот же самый ре-

зультат, что и данное дерево, но имеющее минимальное число старших вершин и требующее минимальное число регистров без обращения к памяти.

Лемма доказывается индукцией по длине максимального пути в дереве T .

Теорема 14. *Алгоритм с учетом ассоциативности порождает оптимальную последовательность операций, вычисляющую данное выражение, если в выражении встречаются только коммутативные, некоммутативные и АК-операторы и никаких других нетривиальных отношений между операторами не допускается.*

В самом деле, минимальность обращений к памяти вытекает из леммы 19. Минимальность засылок в регистры вытекает из теоремы 13. Число бинарных операций, специфицированных в выражении, не изменяется от применения ассоциативных и коммутативных преобразований, поэтому последовательность операций оптимальна.

2.7. Сведение задачи распределения памяти при составлении программ к задаче раскраски вершин графа. Рассмотрим некоторую операторную схему. Каждый оператор может иметь некоторое количество входов и выходов. Входам и выходам сопоставляются величины, называемые *аргументами*, если они сопоставлены входам, и *результатами*, если они сопоставлены выходам. Величина характеризуется *весом*, который представляет собой целое положительное число, указывающее на объем памяти, требуемой для хранения этой величины. Одна и та же величина может быть сопоставлена нескольким выходам одного оператора. Оператор S *вырабатывает величину* x , если x является результатом оператора S , и *воспринимает* x , если x является аргументом оператора S . Величина с весом p , большим единицы, рассматривается как массив, состоящий из p компонент. Для каждого оператора, вырабатывающего некоторую величину-массив x , известно, вырабатывается ли данная величина в целом, т. е. все ее компоненты, или же частично, т. е. отдельная компонента или частичная их совокупность. Появление x в качестве аргумента или результата некоторого оператора называется *вхождением* $E(x)$ величины x в операторную схему. Для любого вхождения $E(x)$ величины x с весом p может быть задано разбиение $E(x)$ на v , $v \in [1:p]$, подвеличин, состоящее в разбиении p на v слагаемых p_i , $i \in [1:v]$, и во введении v величин x_i ,

с весом p_i с указанием их принадлежности к вхождению $E(x)$. Сумму весов различных величин, входящих в схему, назовем *весом схемы*. Следовательно, вес схемы есть объем требуемой для реализации схема оперативной памяти. При условии ее ограниченности возникает задача минимизации занимаемого объема или, что то же, веса схемы.

Назовем *маршрутом* $M(x)$ величины x путь в управляющем графе операторной схемы из оператора S , вырабатывающего величину x , в оператор S_i , воспринимающий величину x , причем ни один из промежуточных операторов маршрута не вырабатывает величину x в целом. Для фиксированных операторов S_i и S_j (начального и конечного) в управляющем графе в общем случае существует множество маршрутов. Так как понятие маршрута включает в себя не только путь в управляющем графе, но и информационную связь, которую этот путь реализует, то каждому множеству маршрутов некоторой величины с фиксированными началом и концом соответствует дуга информационного графа $I(G)$ операторной схемы. Информационный граф есть ориентированный двудольный граф, состоящий из нескольких компонент (односторонней) связности. Каждая такая компонента есть совокупность маршрутов одной и той же величины и носит название *области действия* этой величины. Две области действия *несовместимы*, если в каждой из них найдется информационная связь, причем начальный оператор некоторого маршрута одной связи окажется начальным или внутренним оператором некоторого маршрута другой связи.

Пусть d — некоторая область действия в информационном графе схемы G , $R(d)$ — множество всех операторов схемы, имеющих своими результатами вершины из d , $T(d)$ — множество всех операторов, каждый из которых является внутренним хотя бы для одного из маршрутов информационных связей из d .

Теорема 15. *Две области действия d_1 и d_2 информационного графа схемы G несовместимы тогда и только тогда, когда не пусто множество $R(d_1) \cap R(d_2) \cup \cup R(d_1) \cap T(d_2) \cup R(d_2) \cap T(d_2)$.*

Сопоставим схеме G некоторый граф, называемый *графом несовместимости*, построение которого проводится по следующему правилу. Для каждой области действия d величины с весом p строится полный p -вер-

шинник K_p . Затем для любой пары несовместимых областей действия все вершины соответствующих полных графов соединяются друг с другом.

Правильная раскраска графа несовместимости в минимальное число цветов дает возможность самого экономного распределения памяти. Число использованных цветов дает минимальное необходимое для реализации схемы количество ячеек памяти; при этом всем вершинам графа несовместимости, окрашенным в один цвет, сопоставляется одна и та же величина.

2.8. Сведение задачи о распределении регистров к задаче о раскраске графов. Для системного программиста удобно считать, что в его распоряжении находится неограниченное число регистров, но, прежде чем осуществлять генерацию окончательного кода, необходимо отобразить это неограниченное множество виртуальных регистров на конечное множество регистров реальной машины. Первым шагом в распределении регистров является подготовка графа несовместимости. Каждый виртуальный регистр, так же как и реальный, представляется вершиной графа. Два виртуальных регистра, которые не могут сосуществовать в одном реальном, называются *несовместимыми*; соответствующие вершины в графе несовместимости соединяются ребром. Затем отыскивается минимальная раскраска графа несовместимости. Если потребовавшееся число цветов не превышает числа реальных регистров, то распределение регистров осуществляется закреплением за каждым виртуальным регистром реального регистра того же цвета.

Построение графа несовместимости в данном случае отличается от построения в п. 2.7 отсутствием требования минимальности числа цветов. Будем говорить, что величина x *живет* в точке программы, если в управляющем графе существует путь из входа в вершину, соответствующую оператору, вырабатывающему величину x , далее в вершину P и, наконец, в вершину S_i , где величина x используется, причем нигде на этом пути величина x не перевычисляется. Очевидно, что две величины, живущие одновременно в одной точке программы, несовместимы. Если в некоторой точке программы живут одновременно k величин, то мы должны были бы эту точку программы изобразить полным k -вершинником, что сделало бы хроматическое число графа несовместимости не меньше k . Это не обязательно

но, и достаточно каждой точке программы, в которой живут одновременно k величин x_i и определяется еще одна величина x , сопоставить звезду из k ребер вида (x_i, x) . Если хроматическое число графа несовместимо-сти больше, чем число реальных регистров, требуется перестройка программы для получения нужного хроматического числа.

2.9. Контроль записей в файлах. Пусть в файле хранится некоторая информация в виде записей. Построим граф файловой структуры, взяв в качестве вершин записи, а в качестве дуг — пары (u, v) такие, что запись u содержит указатель на адрес записи v . При этом возникает задача об определении наименьшего числа записей, позволяющих проследить все записи в файле, т. е. все дуги графа файловой структуры. В терминах теории графов эта задача сводится к задаче отыскания наименьшего числа путей, не имеющих общих дуг и покрывающих все дуги орграфа.

2.10. Разбиение программы на участки, удовлетворяющие тому или иному критерию оптимальности, носит название *сегментации* программы, а сами участки называются *сегментами*. Необходимость в сегментации возникает в основном по двум причинам: в связи с использованием многоуровневой памяти и в связи с организацией некоторых оптимизирующих преобразований логической или информационной структуры программы.

Сегментация первого типа проводится тогда, когда программа не помещается в выделенную ей память высокого уровня. Тогда приходится хранить в ней лишь активный в данный момент фрагмент программы и вызывать другие фрагменты из памяти низкого уровня по мере обращения к ним. На такого рода обмены между оперативной и внешней памятью и сопутствующие прерывания тратится немалое машинное время, поэтому, даже если есть возможность совмещения их с основным счетом, пользователь заинтересован в том, чтобы они производились как можно реже. Отсюда вытекает специфическое для этого вида сегментации требование: минимизация числа обращений между сегментами.

Необходимость в сегментации второго рода возникает тогда, когда имеется некоторая система оптимизирующих преобразований логико-информационной структуры программы, для применения которой нужно представление программы в более простом и унифици-

рованном виде. Эта система может быть набором формальных преобразований, приводящих к экономии памяти или операторов программы, уменьшению избыточности вычисления, либо состоять из алгоритмов статистического или динамического распараллеливания.

Рассмотрим один подход к сегментации первого рода. Главным принципом сегментации программ с учетом информационных связей является объединение в сегменты операторов естественных частей программы, т. е. линейных участков и правильных циклов; если такое разрезание не удовлетворяет ограничениям на объем, производится дальнейшая сегментация, при которой могут минимизироваться связи между сегментами.

Если выделенные естественные части программы удовлетворяют заданным ограничениям на объем информации (командной и числовой), то процесс разбиения на этом заканчивается. Если же некоторая естественная часть, выделенная при просмотре исходной программы, окажется недопустимой по объему, то необходимо разрезать ее на части. Это разрезание можно производить на основе алгоритма выделения цепочек в схеме программы, относящейся к данной естественной части. Под *цепочкой* здесь подразумевается последовательность функционально связанных операторов, начинающаяся с оператора, функционально не зависящего ни от какого оператора естественной части, и заканчивающаяся в операторе, который функционально не управляет никакими другими операторами естественной части. Выделение цепочек основано на анализе матрицы $F = \|f_{ij}\|$ функциональной зависимости:

$$f_{ij} = \begin{cases} 1, & \text{если } i < j \text{ и результат оператора с номером } i \text{ используется в качестве аргумента в операторе с номером } j; \\ 0 & \text{в противном случае.} \end{cases}$$

Способ перечисления цепочек аналогичен построению всех путей между заданными вершинами в графе, матрица смежности которого есть F .

2.11. Крупноблочная сегментация и распараллеливание схем программ. С увеличением быстродействия вычислительных средств стала необходимой разработка новых методов группировки операторов программы. Дело в том, что в современных многопроцессорных вы-

числительных системах скорость обработки данных становится сравнимой со скоростью анализа структуры программы. Назначение оператора на работу или выделение отдельной параллельной ветви, равно как и некоторое формальное преобразование, требующее для своего выполнения определенной подготовительной работы, становятся сравнительно медленными и дорогостоящими актами. Поэтому желательно, чтобы каждый оператор анализируемой схемы представлял достаточно крупное задание, например вычисление процедуры или подпрограммы. Такое естественное разбиение программы не всегда возможно, приходится разбивать программу на некоторые произвольные части. Учитывая то, что разветвленная программа с трудом поддается распараллеливанию, кроме требования минимизации информационной взаимосвязи сегментов к этому разбиению предъявляется свое специфическое требование: результирующая структура, элементарным объектом которой является сегмент, должна иметь минимальное число разветвлений.

Пусть дана операторная схема над общей памятью с управляющим графом G , множеством операторов A , причем для каждого оператора $a \in A$ определены конечное множество входных переменных $In(a)$ и конечное множество выходных переменных $Out(a)$. Пусть дано некоторое разбиение множества операторов на непересекающиеся сегменты — *сегментация* σ . *Фактор-схемой* \bar{S} по сегментации σ называется операторная схема, операторами которой являются элементы множества сегментов \bar{A} , для каждого из которых определены множества $In(\alpha)$ и $Out(\alpha)$, $\alpha \in \bar{A}$; весом сегмента $w(\alpha)$ является сумма весов входящих в него исходных операторов. В этих обозначениях величина

$$\sum_{\alpha, \beta \in \bar{A}} |In(\alpha) \cap Out(\beta)|$$

определяет сумму межсегментных информационных связей. Величина

$$f(\bar{S}) = \frac{1}{n_{cp}} \sum_{\alpha, \beta \in \bar{A}} |In(\alpha) \cap Out(\beta)|,$$

где n_{cp} — средняя длина естественной части схемы S , называется *коэффициентом информационной схемы* \bar{S} .

Назовем *полугамаком* сегмент, все выходящие дуги которого ведут в одну вершину. Ясно, что любой гамак является полугамаком. Если в схеме выделен полугамак, то путем дублирования его для каждой входящей дуги можно получить эквивалентную схему, в которой выделенными сегментами являются уже гамаки.

Основная цель алгоритма крупноблочной сегментации состоит в уменьшении коэффициента информативности $f(\bar{S})$ путем «съедания» при объединении в сегменты информационных связей и увеличения средней длины естественной части схемы n_{cp} .

Вход: Управляющий граф G , ограничения на размер сегмента сверху M и снизу m .

Выход: Множество допустимых сегментов $\{ \}$.

начало

1. выделить линейные компоненты α в G ;

2. для всех линейных компонент α цикл

3. **начало**

4. если $w(\alpha) > M$ то

5. **начало**

6. выделить в α гамаки β с $w(\beta) \leq M$;

7. если $w(\beta) < m$ то

8. построить гамак β до полугамака, удовлетворяющего условиям $m \leq w(\beta) \leq M$;

9. **конец**

10. **конец** цикла;

11. сегментировать оставшуюся часть путем попарного слияния вершин графа G ;

конец

§ 3. Применение деревьев для организации больших массивов информации

Если таблица ключей (словарь) динамически изменяется, т. е. ключи постоянно добавляются и удаляются, то экономия от применения бинарного поиска (поиск методом деления пополам) не покроет расходов на поддержание упорядоченного расположения ключей. Выход из этой ситуации состоит в использовании вместо таблицы структуры бинарного или m -арного дерева, которая позволяет быстро вставлять и удалять ключи и производить эффективный поиск нужного ключа.

3.1. Балансированные по высоте бинарные деревья (АВЛ-деревья). При определении дерева сортировки в гл. 1 говорилось, что для произвольно взятого множест-

ва слов дерево может расти крайне неравномерно и легко представить себе ситуацию, когда дерево вырождается в линейный список. Это объясняется тем, что структура дерева практически полностью определяется порядком, в котором слова подаются на вход алгоритма.

Пример. На рис. 6.14, а показано бинарное дерево над множеством названий дней недели, взятых в их естественном порядке. Для сравнения на рис. 6.14, б показано дерево над тем же множеством слов, но взятых в алфавитном порядке.

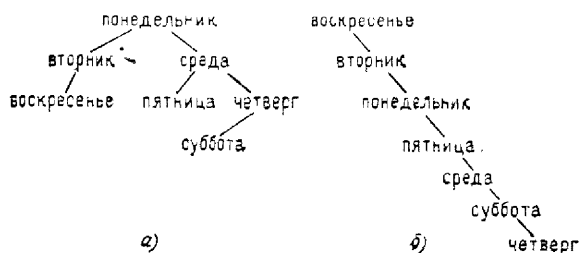


Рис. 6.14.

Основными операциями при работе с динамическим словарем являются поиск, включение и удаление слова. В отличие от линейно упорядоченного массива, где эти операции имеют трудоемкость, пропорциональную длине массива, в словаре, организованном в виде АВЛ-дерева, указанные операции имеют логарифмическую трудоемкость. Для сохранения ее при добавлении или удалении слова требуются специальные усилия.

В АВЛ-дереве, где критерием оптимальности служит степень балансированности дерева по высоте, трудоемкость поиска пропорциональна высоте дерева. Таким образом, трудоемкость поиска в худшем случае не превосходит $(3/2) \log n$, где n — количество слов в словаре или, что то же, вершин в АВЛ-дереве. Восстановление структуры АВЛ-дерева при включении нового слова или удалении производится операциями вращения и двойного вращения. Последовательность построения АВЛ-дерева для множества названий дней недели, взятых в лексикографическом порядке, показана на рис. 6.15.

Включение и удаление слов в АВЛ-дереве требуют движения по дереву от корня до искомой вершины

(в случае включения — до места, где может быть подвешена новая вершина) и обратного движения по тому же пути, что требует стекового механизма. Кроме того, требуется два бита памяти для каждой вершины для указания, какое из трех возможных соотношений между высотами ее двух поддеревьев имеет место.

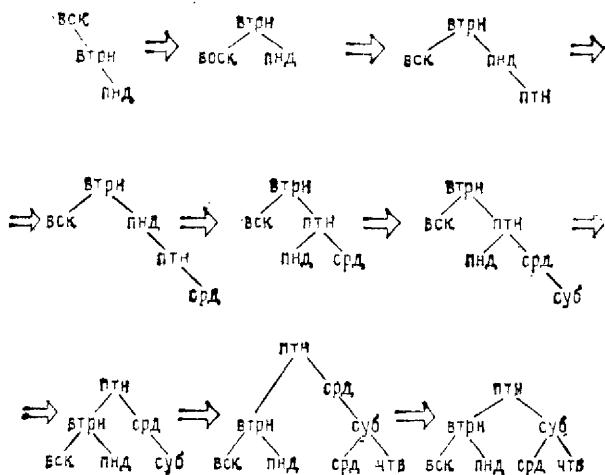
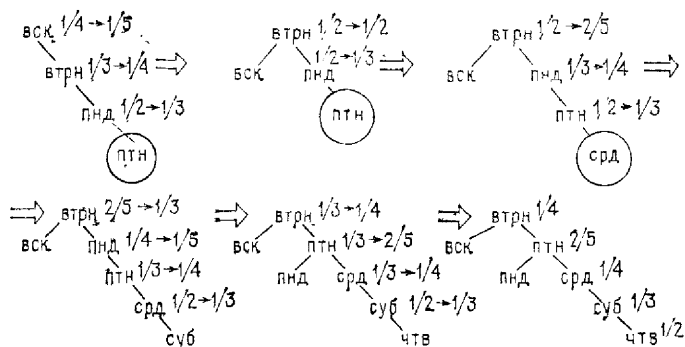


Рис. 6.15.

Можно обобщить понятие АВЛ-дерева, допустив, чтобы высоты правого и левого поддеревьев любой вершины не отличались больше чем на $\Delta > 1$. Рост дисбаланса вызывает увеличение среднего времени поиска по сравнению со случаем $\Delta = 1$, причем это увеличение не зависит асимптотически от размера дерева и приблизительно равно $0,28\Delta$. Но эта тенденция становится заметной только для больших Δ . Аналогичные исследования относительно среднего числа перестроек дерева показали, что число перестроек падает с ростом Δ и похоже, что $\Delta = 5$ минимизирует полное время процедур перестройки в АВЛ-дереве. Корректный выбор Δ зависит от относительной частоты включения, удаления или чистого поиска, так как дерево, преимущественно используемое для поиска, должно быть АВЛ-деревом, т. е. иметь $\Delta = 1$.

3.2. Балансированные по весу бинарные деревья (ВВ-деревья). ВВ-деревья не получили столь широкого распространения, как АВЛ-деревья. Их преимущества

перед АВЛ-деревьями состоит в реализации включения и удаления слов однократным движением вниз от корня и в возможности установления желательных соотношений между временами поиска и включения/удаления подходящим выбором баланса α .



не хранится. Пример построения *ВВ*-дерева ($\alpha = 1/4$) над множеством названий дней недели, взятых в лексикографическом порядке, дан на рис. 6.16.

в виде какого-либо варианта самоорганизующегося дерева, если множество слов заранее не известно и речь, следовательно, идет о динамическом словаре.

3.4. Выровненные деревья. В отличие от АВЛ- и ВВ-деревьев, где информация хранится во внутренних вершинах, в выровненных бинарных деревьях информация хранится в висячих вершинах, а во всех остальных вершинах хранится вспомогательная информация, необходимая для работы с деревьями. Чаще всего во внутренней вершине хранится наибольшее слово из имеющихся в этом поддереве слов. При такой организации вспомогательной информации процедура построения выровненного дерева также основывается на движении к левому потомку, если включаемое слово меньше слова в левом потомке рассматриваемой вершины, и к правому потомку в противном случае. Получаемое при этом выровненное дерево является упорядоченным в том смысле, что информация в листьях упорядочена слева направо.

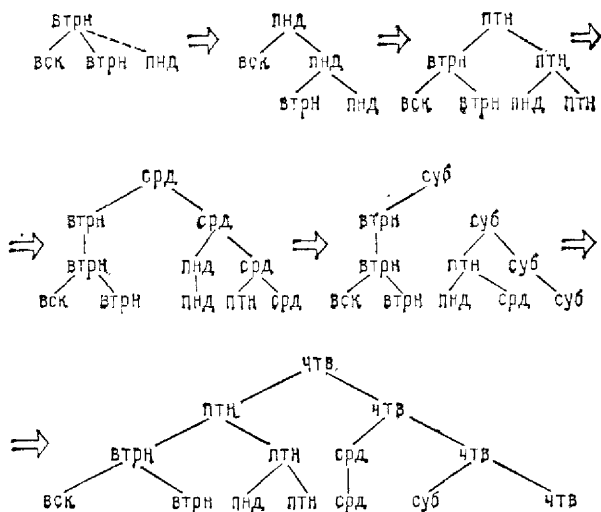


Рис. 6.17.

Пример построения *H*-дерева над множеством названий дней недели показан на рис. 6.17, а пример построения *HВ*-дерева над тем же множеством показан на рис. 6.18.

3.5. В-деревья. В случае очень больших массивов информации наиболее реалистичным является предположение, что большая часть поискового дерева должна располагаться во внешней памяти, откуда части дерева пересылаются по мере необходимости в основную память. Устройства внешней памяти (барабаны, диски и т. д.) имеют более затрудненный доступ и большее время ожидания, но высокую скорость пересылки данных после начала пересылки. Таким образом, если дерево размещено правильно, то достаточно большие его части будут пересылаться в основную память блоками, и скорость работы с деревом будет определяться не числом сравнений, необходимых для поиска слова, а средним числом пересылаемых блоков.

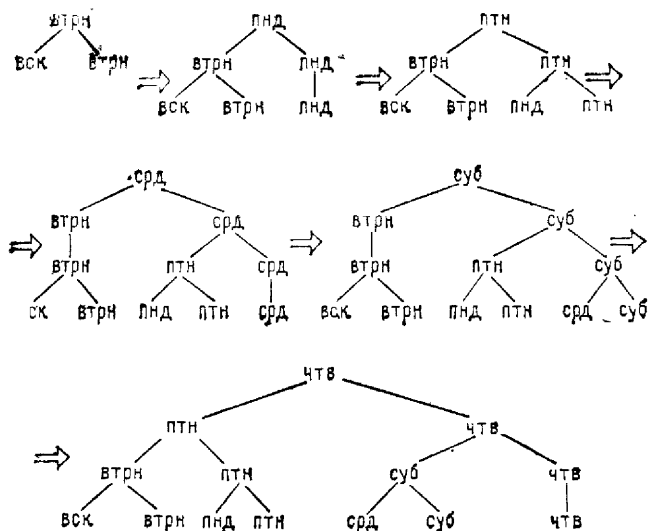


Рис. 6.18.

Наибольшее значение для работы с двухуровневой памятью имеют В-деревья и их модификации.

В-деревом порядка m называется m -арное выровненное дерево, у которого:

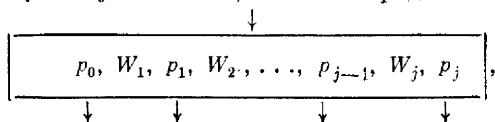
а) ни одна из висячих вершин не содержит информации;

б) каждая невисячая вершина с k потомками содержит $k - 1$ слов, $m/2 \leq k \leq m$ (для корня $k \leq 2$);

в) каждая невисячая вершина содержит k указателей на своих потомков.

Замечание. При графическом изображении висячие вершины B -дерева обычно не рисуют.

Из определения видно, что вершину, содержащую j слов и $j+1$ указателей, можно представить в виде



где $W_1 < W_2 < \dots < W_j$, а указатель p_i указывает на поддерево, в котором хранятся слова, больше W_i и меньше W_{i+1} .

На рис. 6.19 показано B -дерево порядка 5 над множеством слов $\{A, B, \dots, Y\}$.

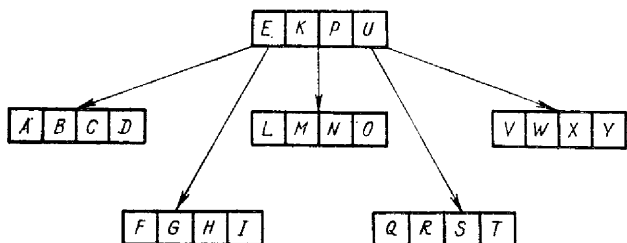


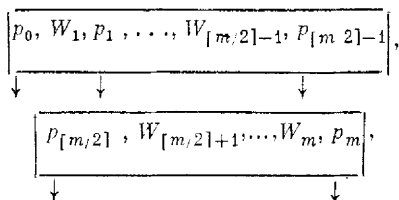
Рис. 6.19.

B -дерево порядка 3 носит отдельное название *2-3-дерева*.

Поиск в B -дереве организуется следующим образом. Вначале в оперативную память засылается корень дерева и в нем ищется требуемое слово; если его там нет, то в оперативную память вызывается вершина, на которую указывает указатель p_i такой, что $W_i < W < W_{i+1}$ (W — искомое слово). Указатель p_0 используется, если требуемое слово меньше W_1 , а указатель p_j используется, если требуемое слово больше W_j . Если соответствующий указатель на некотором шаге поиска указывает на висячую вершину, то поиск неудачен.

Включение нового слова производится так. Вначале производится поиск включаемого слова W . Если слово W уже находится в дереве, то поиск заканчивается удачно и алгоритм заканчивается. В противном случае поиск неудачен и заканчивается в вершине, у которой все потомки — висячие вершины, новое слово W вклю-

чают в эту вершину, соблюдая лексикографический порядок и добавляя еще один указатель и висячую вершину. Если общее число слов в вершине после этого не превосходит $m - 1$ для дерева порядка m , то процедура включения слова заканчивается. Если же в вершине стало m слов, то ее расщепляют на две вершины:



а слово $W_{[m/2]}$ помещают в вершину-предка. Таким образом, указатель p в вершине-предке заменяется на последовательность $p, W_{[m/2]}, p'$, что в свою очередь может привести к расщеплению. Если нужно расщепить корень дерева, который не имеет предка, то просто создают новый корень и помещают в него единственное слово $W_{[m/2]}$, в таком случае дерево становится на единицу выше. Данная процедура включения слов сохраняет все свойства B -дерева.

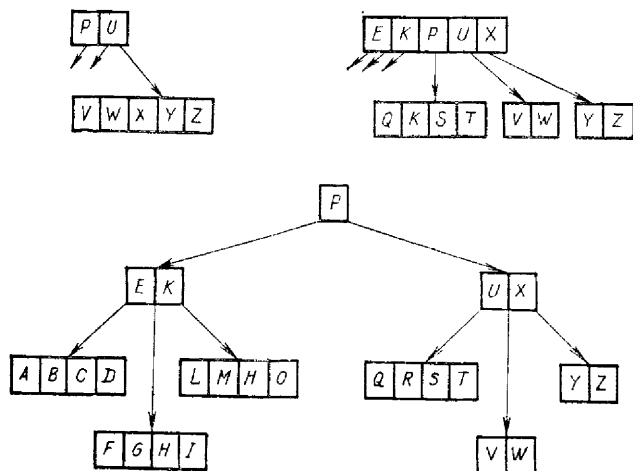
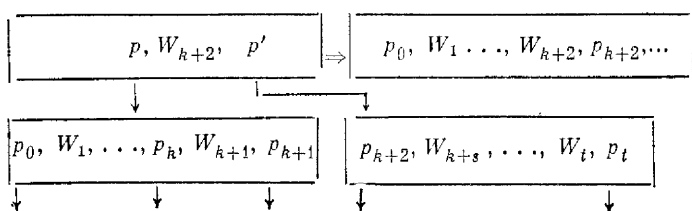


Рис. 6.20.

Пример включения слова Z в дерево, изображенное на рис. 6.19, показан на рис. 6.20.

Для удаления слова W вначале проведем поиск. Если слово W находится в вершине самого нижнего уровня и после его исключения вершина будет содержать не меньше допустимого числа слов, то слово W просто исключается из соответствующей вершины. Если при этом в вершине остается слов меньше допустимого числа, то вершина сливается с одним из своих братьев (либо с ближайшим слева, либо с ближайшим справа) по правилу



Если получившаяся после слияния вершина оказывается переполненной, то она снова расщепляется так, как это делается при включении нового слова. Так как при слиянии двух вершин слово-разделитель из вершины-предка удаляется, то может возникнуть ситуация недозаполненности вершины. В этом случае поступаем так, как это было описано выше.

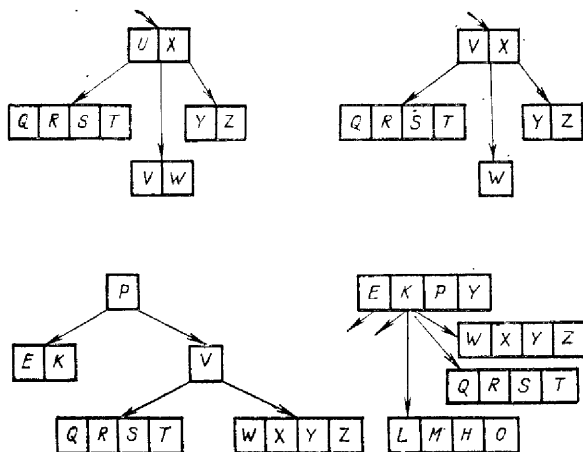


Рис. 6.21.

Пример удаления слова U из дерева, показанного на рис. 6.20, приведен на рис. 6.21.

3.6. Плотные m -арные деревья. Недостатком B -деревьев является слабая заполняемость их вершин словами. Увеличить заполняемость можно как с помощью перехода к деревьям, у которых каждая невисячая вершина, отличная от корня, имеет не менее $\lfloor (2m - 1)/3 \rfloor$ потомков, а корень — не менее двух и не более m потомков (B^* -деревья), так и за счет использования плотных m -арных деревьев.

Способ хранения информации в плотном m -арном дереве аналогичен способу хранения ее в B -деревьях. Каждое плотное m -арное дерево над множеством из n слов имеет $n + 1$ висячих вершин. Индукцией по высоте дерева h легко показать с использованием рекуррентной формулы (4.34), что для каждого r -плотного m -арного дерева высоты h имеет место оценка

$$L_{\min}(r, m, h) \geq a_h > m^{\lfloor n/2 \rfloor}.$$

Следовательно, каждое r -плотное m -арное дерево над множеством из n слов есть дерево высоты $h \leq \leq 2 \log_m(n + 1)$. Отсюда следует, что r -плотное m -арное дерево не может вырождаться в линейный список. (Заметим, что приведенная оценка очень груба).

Чтобы включить новое слово W в плотное m -арное дерево T , вначале осуществим поиск слова W в дереве T . Поскольку он неудачен (слова W в дереве нет), то поиск заканчивается в висячей вершине. Если ее предок не насыщен, то слово W включается в вершину-предка v , в противном случае требуется применение громоздких процедур восстановления структуры дерева.

3.7. Организация поиска слов внутри вершин B -дерева. Так как порядок m B -дерева и тем самым количество слов, хранимых в одной вершине, не ограничивается сверху, можно представить себе ситуацию, когда количество слов будет составлять несколько сотен. В таких случаях нужна организация эффективного поиска слов внутри вершин. Для этого массив слов в вершине можно организовать в виде списков, списков с двойными связями или в виде балансированного дерева. Соответствующие B -деревья будем называть B - S -, B - L - или B - T -деревьями. Последний тип допускает обобщение на случай, когда структура данных в виде B - T -дерева используется одновременно несколькими пользователями.

Мы знаем, что когда вершина в B -дереве порядка m становится переполненной (в ней оказывается

m слов), она распадается на две, каждая из которых имеет не менее $\lfloor m/2 \rfloor - 1$ слов. В B - S -дереве с последовательным расположением слов в вершине разбиение может быть произведено с трудоемкостью $O(m)$ обычными способами. Когда же вершина организована как балансированное дерево, процесс разделения вершины уже не так прост, как бы хотелось, хотя и имеет ту же трудоемкость. Это приводит к необходимости обобщить понятие B -дерева, ослабив требование относительно минимального числа слов в вершине и добившись этим облегчения разделения вершины на две.

Пусть $f: N \rightarrow N$ — функция, заданная на множестве натуральных чисел, такая, что для всех $m \geq 3$ справедливо неравенство $2 \leq f(m) \leq \lfloor m/2 \rfloor$. $f(m)$ -деревом называется страничное дерево (т. е. дерево, у которого каждая вершина соответствует странице памяти), обладающее следующими свойствами:

1. Каждая вершина имеет не более m потомков.
2. Каждая вершина, исключая корень и висячие вершины, имеет по крайней мере $f(m)$ потомков.
3. Корень имеет не менее двух потомков.
4. Все висячие вершины располагаются на одном и том же уровне и не несут информации.
5. Невисячая вершина с $j+1$ потомками содержит j слов и может быть изображена в виде, аналогичном виду вершины в B -дереве.

Как видно из этого определения, в отличие от B -дерева в $f(m)$ -дереве минимальное число потомков у вершины может изменяться от двух до $\lfloor m/2 \rfloor$ в зависимости от функции f . Если функция f такова, что $f(m) = \lfloor m/2 \rfloor$, мы получаем обычное B -дерево порядка m . Для монотонно возрастающей функции f класс $f(m)$ -деревьев ведет себя очень хорошо: минимальное число слов в невисячей вершине растет вместе с ростом параметра m .

Поиск, включение и удаление слов в $f(m)$ -деревьях практически не отличаются от аналогичных алгоритмов для B -деревьев.

Для включения слова W , как обычно, начинаем с поиска вводимого слова W в $f(m)$ -дереве. Если поиск успешен, то слово W уже в дереве и процесс заканчивается. В противном случае слово W включается в соответствующую вершину — предка висячей вершины, где закончился неуспешный поиск. Если в этой вершине содержится менее m слов, включение нового

слова завершено. Если она становится насыщенной с m словами и имеет форму

$$v: \left[p_0, W_1, p_1, \dots, W_{s-1}, p_{s-1}, W_s, p_s, W_{s+1}, p_{s+1}, \dots, W_m, p_m \right],$$

$\downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow$

то она распадается на две вершины: вершину v' , имеющую вид

$$\begin{array}{c} \downarrow p \\ \left[p_0, W_1, p_1, \dots, W_{s-1}, p_{s-1} \right], \end{array}$$

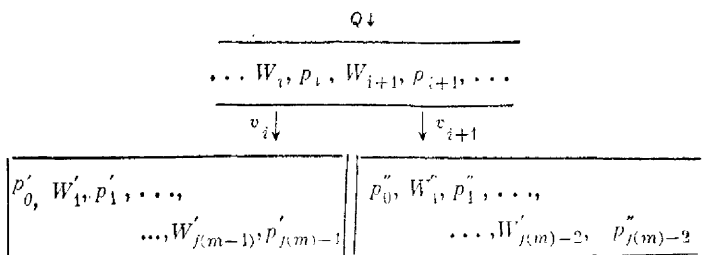
и вершину v'' , имеющую вид

$$\begin{array}{c} \downarrow p_1 \\ \left[p_s, W_{s+1}, p_{s+1}, \dots, W_m, p_m \right], \end{array}$$

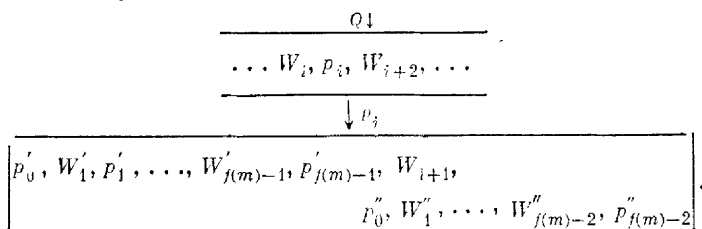
соответственно с $s-1$ и $m-s$ словами, где $s \equiv [f(m) : [m/2]]$. Слово W_s включается с указателем p' в вершину — предка вершины v . Может случиться, что предок вершины v сам станет насыщенным, тогда процесс распада вершин продолжается далее. Так как $2 \leq f(m) \leq s \leq [m/2]$, то мы имеем $f(m) \leq m-s+1 \leq m$ и число слов в вершине v' , появившейся в результате распада вершин, находится внутри интервала $(f(m)-1, m-1)$. Ясно, что все свойства $f(m)$ -деревьев при включении слова сохраняются.

При удалении слова мы также начинаем с поиска удаляемого слова W . Если оно не найдено, то процесс заканчивается. Если оно расположено в вершине — предке висячей вершины, то мы просто удаляем его. В противном случае оно заменяется наименьшим словом из поддерева, на которое указывает указатель p_i , соответствующий удаляемому слову $W = W_i$. Слово, заменившее слово W_i , в свою очередь удаляется из вершины, в которой оно было первоначально расположено. В любом случае мы приходим к удалению слова из вершины — предка висячей вершины. Если после удаления она содержит не менее $f(m)$ слов, то процесс заканчивается. В противном случае мы должны применить либо перестановку слов, либо слияние вершин. Если смежный брат вершины имеет не менее $f(m)$ слов, то слова в этих двух вершинах вместе с разделя-

ющим словом в их предке могут быть переставлены так, что каждая вершина будет иметь не менее $f(m) - 1$ слов. Если не имеет, то тогда



В этой ситуации мы просто сливаем вершины v_i и v_{i+1} вместе с разделяющим словом W_{i+1} в вершине — предке Q и удаляем пару (W_{i+1}, p_{i+1}) из вершины Q ; при этом получаем



При удалении пары (W_{i+1}, p_{i+1}) из вершины Q может случиться так, что вершина Q теперь имеет меньше $f(m) - 1$ слов, и этот процесс должен быть продолжен. Заметим, что число слов в вершине v_i после слияния равно $2f(m) - 2$ и неравенство $f(m) \leq 2f(m) - 1 \leq m$ непосредственно следует из неравенства $2 \leq f(m) \leq \lfloor m/2 \rfloor$. Очевидно, что удаление слова преобразует $f(m)$ -деревья в $f(m)$ -деревья с сохранением всех их свойств.

Теорема 16. Пусть $f: N \rightarrow N$ есть функция, для которой при $m \geq 3$ выполнено неравенство $2 \leq f(m) \leq \lfloor m/2 \rfloor$. Тогда:

1) поиск в $f(m)$ -дереве, содержащем n слов, имеет трудоемкость самое большее $O(\log_{f(m)} n)$;

2) включение и удаление слов в $f(m)$ -дереве, содержащем n слов, при которых получающееся в результате дерево сохраняет структуру $f(m)$ -дерева, имеют трудоемкость $O(\log_{f(m)} n)$.

Следует заметить, что эти алгоритмы сводятся к алгоритмам для B -деревьев порядка m , когда $f(m) = \lfloor m/2 \rfloor$. Однако они работают для любой функции f , удовлетворяющей условию $2 \leq f(m) \leq \lfloor m/2 \rfloor$. Важным свойством $f(m)$ -деревьев является то, что при расщеплении вершины нет необходимости делить ее точно пополам, как это делается в B -деревьях. Вместо этого мы выбираем точку раздела между $f(m)$ и $\lfloor m/2 \rfloor$, причем $f(m)$ может быть равной двум.

В рассмотренных алгоритмах использовались элементарные операции:

- а) ИСКАТЬ слово в вершине.
 - б) ВКЛЮЧИТЬ слово в вершину после выявления его места в ней.
 - в) УДАЛИТЬ слово из вершины после выявления его места в ней.
 - г) РАСЩЕПИТЬ насыщенную вершину на две.
 - д) СЛИТЬ
 - е) ПЕРЕСТАВИТЬ
- } слова в двух смежных вершинах вместе с разделяющим словом в их предке.

Для поиска слова используется операция а), для включения слова — операции а), б) и г), для удаления — операции а), в), д) и е). Заметим, что первые три операции относятся к внутривершинным операциям, а остальные — к межвершинным. Оценим различные способы организации вершинных массивов слов с точки зрения этих элементарных операций.

S -деревом (L -деревом) порядка m называется $f(m)$ -дерево, в котором все вершинные массивы слов организованы как последовательный список (список с двойными связями). Аналогично определяются B - S - и B - L -деревья.

В S -деревьях операция ИСКАТЬ может быть осуществлена за время $O(\log m)$, тогда как операции ВКЛЮЧИТЬ и УДАЛИТЬ требуют времени $O(m)$. Для L -деревьев операции ВКЛЮЧИТЬ и УДАЛИТЬ требуют времени, равного константе, тогда как операция ИСКАТЬ требует $O(m)$ времени. В любом случае внутривершинные операции требуют $O(m)$ времени, так как, прежде чем включить или удалить слово, мы применяем операцию ИСКАТЬ. Операция ПЕРЕСТАВИТЬ — более простая операция и требует $O(m)$ времени в S -деревьях и постоянного времени в L -деревьях. Операции СЛИТЬ и РАСЩЕПИТЬ не содержат ни-

чего, кроме простой пересылки слов из одной вершины в другую, и поэтому требуют $O(m)$ времени и для S - и для L -деревьев.

Трудоёмкость $O(m)$ для внутривершинных операций является серьёзным недостатком для S - и L -деревьев. Это приводит к мысли использовать для внутренней организации массива слов в вершинах дерева (АВЛ-деревья, 2-3-деревья и т. д.), что приводит к понятию T -дерева.

T -деревом порядка m называется $f(m)$ -дерево, в котором все вершины представляют собой балансированные деревья частного типа. (Аналогично определяется B - T -дерево.)

Очевидно, что поиск, включение и удаление слов в T -дереве те же самые, что и для $f(m)$ -деревьев, за исключением того, что мы должны точно объяснить, как работают элементарные операции а) — е).

Для операций ИСКАТЬ, ВКЛЮЧИТЬ и УДАЛИТЬ могут быть использованы стандартные алгоритмы для выбранного типа балансированных деревьев. Они требуют в худшем случае $O(\log m)$ времени. Операции ПЕРЕСТАВИТЬ и СЛИТЬ — более простые операции: операция ПЕРЕСТАВИТЬ может быть сделана за время $O(\log m)$; при выполнении операции СЛИТЬ мы вначале пересылаем слова из вершины в ее смежного брата, что требует $O(m)$ времени, а потом производим слияние двух балансированных деревьев и включение разделительного слова из вершины, являющейся предком обоих деревьев, за время $O(\log m)$. Все это даст оценку $O(m)$.

РАСЩЕПИТЬ — самая трудная из этих элементарных операций. Одним из способов выполнить эту операцию является следующий:

1. Выделить разделяющее слово W , (медиану в случае B - T -дерева). Это можно сделать за время $O(m)$.

2. Расщепить балансированное дерево на два балансированных дерева слева и справа от W . Это требует $O(\log m)$ времени.

3. Скопировать одно поддереву в новую вершину за время $O(m)$.

И хотя с помощью более совершенной техники нахождение разделяющего слова можно довести до операции с трудоёмкостью $O(\log m)$, полная трудоёмкость операции РАСЩЕПИТЬ остается равной $O(m)$.

Основываясь на анализе элементарных операций, можно заключить, что T -деревья лучше подходят для организации больших массивов информации, нежели S -, L - и, в частности, B -деревья.

Библиографический комментарий

Большинство примеров применения перечисленных в таблице теоретико-графовых задач дано в тех же статьях, где эти задачи были описаны. Ссылки на эти статьи даны в соответствующих главах. Таблицу, естественно, можно было бы продолжить. Например, можно было бы указать на сведение задачи о проверке правильности программ к задаче построения усиленной базы орграфов [23].

Пути i -го уровня рассматривались в работе М. Пэйджа [143], методы вычисления времени исполнения программ — в работе В. И. Знака [32]; ссылки на другие работы можно найти в обзоре [19]. Операции над управляющими графами изучались в работе [101], структурированные классы $D_1 — D_4$ — в работах В. Э. Иткина [37, 38], эквивалентные преобразования управляющих графов — в работах [98, 107].

Цикломатическая сложность программ была введена Т. Мак-Кэйбом в работе [133], в настоящее время известен ряд ее модификаций. О двух из них см. [111, 137]. С различными стратегиями тестирования, в том числе основанными на цикломатической сложности, можно познакомиться в работах школы В. В. Липаева, например в [56, 57, 65]. Там же можно найти ссылки на другую литературу. Задача о контрольных точках излагается по работе [77].

Сведение задачи о размещении блоков программы в памяти к задаче о коммивояжере рассматривалось в заметке Р. Карпа [40], проблема очередности активизации задач рассматривалась в работе [59], задача о минимизации памяти при вычислении арифметических выражений — в работах [28, 71] (см. также [25]). Р. Сети и Дж. Ульман в работе [148] в 1970 г. рассмотрели задачу минимизации числа программных шагов и/или числа обращений к памяти при вычислении арифметических выражений, если доступно фиксированное число регистров. Предложенные ими алгоритмы существенно используют коммутативные и ассоциативные свойства операций. Они доказали оптимальность алгоритмов, используя различные критерии стоимости, включая длину получаемого на выходе кода. Эти результаты были усилены в 1975 г. А. Ахо и С. Джонсоном [89], которые показали, что, используя динамическое программирование, можно получить оптимальный код за линейное время для более широкого класса машин. Позднее подход Сети и Ульмана получил свое развитие в целом ряде работ.

Сведение задачи об экономии памяти к задаче о раскраске вершин графа рассматривали С. С. Лавров [54] и А. П. Ершов [29]. Подробно эта проблема исследована в работе [30]. Там же описан метод построения информационного графа с помощью транзитивного замыкания относительно множества вершин. Описанный в работе [29] подход был реализован при разработке системы программирования альфа [3]. Сведение к раскраске

графов задачи о распределении регистров рассматривалось в работе [106].

Подход к сегментации программ, изложенный в п. 2.10, принадлежит Д. А. Поспелову [68]. К сожалению, данный алгоритм применим только для разрезания линейных участков. Проблема разрезания естественных частей, представляющих собой циклические участки, более сложна. Для решения ее в [68] указан точный метод, а в [51] — приближенный, являющийся частью общего метода сегментации, который может быть полезен, если программа допускает эффективное разделение по циклам. Укажем еще на метод С. Н. Бушева [43], который основан на методе Д. А. Поспелова и имеет трудоемкость, пропорциональную числу вершин в управляющем графе сегментируемой программы. В методе Бушева на первом этапе производится разрезание на уровне операторов. На следующих этапах операторы склеиваются согласно требованиям, предъявляемым к сегментированию, которые преследуют цель получения оптимального в некотором смысле разрезания. Вопросы крупноблочной сегментации и распараллеливания программ рассматривались в работе [14].

В-деревья были введены Р. Байером и К. Мак-Крэйтом в 1972 г. [100] и подробно исследовались в [48]. Подробнее с 2-3-деревьями можно познакомиться в [7]. В*-деревья были введены Д. Кнудом в [48], они позволяют использовать до $2/3$ имеющегося в каждой вершине пространства. Плотные m -арные деревья основаны на понятии братского дерева и позволяют строить деревья с наперед заданной плотностью заполнения вершин. Этот тип деревьев был предложен в 1979 г. К. Чуликом, Т. Отманом и Д. Вудом [108]. Организация поиска слов внутри вершин В-деревья изучалась в работе [125].

СПИСОК ЛИТЕРАТУРЫ

1. Адельсон-Вельский Г. М., Диниц Е. А., Карзанов А. В. Потокковые алгоритмы.— М.: Наука, 1975.
2. Адельсон-Вельский Г. М., Ландис Е. М. Один алгоритм организации информации.— ДАН СССР, 1962, 146, № 2, с. 263—266.
3. АЛБФА — система автоматизации программирования: Сб. трудов/Под ред. А. П. Ершова.— Новосибирск: Наука, 1967.
4. Анисимов А. В. Об оптимальной упаковке деревьев.— Кибернетика, 1976, № 3, с. 89—91.
5. Арлазаров В. Л., Диниц Е. А., Кронрод М. А., Фараджев И. А. Об экономном построении транзитивного замыкания ориентированного графа.— ДАН СССР, 1970, 194, № 3, с. 487—488.
6. Арлазаров В. Л., Усков А. В., Фараджев И. А. Алгоритм нахождения всех простых циклов в ориентированном графе.— В кн.: Исследования по дискретной математике. М.: Наука, 1973, с. 184—188.
7. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.— М.: Мир, 1979.
8. Бекпшев Г. А. Об одной задаче разбиения вершин ориентированного графа на классы.— В кн.: Дискретный анализ, вып. 6. Новосибирск: Наука, 1966, с. 27—34.
9. Берж К. Теория графов и ее применения.— М.: ИЛ, 1962.
10. Берзтисс А. Т. Структуры данных.— М.: Статистика, 1974.
11. Буда А. О., Сабельфельд В. К. Анализ потока данных и эквивалентность программных машин.— В кн.: Математическая теория и практика систем программного обеспечения. Новосибирск: ВЦ СО АН СССР, 1982, с. 45—63.
12. Бульонкова А. А. Приближенный алгоритм раскраски больших графов.— В кн.: Проблемы теоретического и системного программирования. Новосибирск: Изд-во НГУ, 1982, с. 81—86.
13. Бушев С. Н. Об одном подходе к задаче разрезания программ на функционально независимые части.— Изв. АН СССР. Техническая кибернетика, 1970, № 5, с. 110—112.
14. Вальковский В. А., Касьянов В. Н. Крупноблочная сегментация и распараллеливание схем программ.— Программирование, 1976, № 1, с. 16—26.
15. Венгеров В. Н., Никитин А. С. Алгоритмы разрезания взвешенных графов.— Минск, 1978. (Препринт/ИМ АН БССР: 20 (52)).

16. Геолещян Г. Г. О задаче оптимального размещения вершин графа на отрезке.— ДАН Арм. ССР, 1973, 56, № 5, с. 269—271.
17. Геолещян Г. Г. Плоское размещение дерева с минимизацией длины.— Вопросы радиоэлектроники. Сер. ЭВТ, 1975, вып. 6, с. 87—91.
18. Геолещян Г. Г. Об оптимальных размещениях и их свойствах.— ДАН Арм. ССР, 1975, 61, № 2, с. 65—69.
19. Головкин Б. А. Графовые модели программ с вероятностными параметрами. (Современное состояние).— Вопросы радиоэлектроники. Сер. ЭВТ, 1976, вып. 6, с. 3—26.
20. Голубев-Новожилов Ю. С. Многомашинные комплексы вычислительных средств.— М.: Сов. радио, 1967.
21. Гольдберг М. И., Клишкер И. А. Алгоритм минимальной нумерации вершин дерева.— Сообщ. АН ГССР, 1976, № 3, с. 553—556.
22. Грунский И. С., Сперанский Д. В. О покрытии графа множеством путей, исходящих из одной вершины.— Кибернетика, 1974, № 1, с. 29—34.
23. Дамбит Я. Я. Существование усиленной базы направленного графа и некоторая задача программирования.— Автоматика и вычислительная техника, 1963, № 4, с. 69—79.
24. Диниц Е. А., Зайцев М. А., Карзанов А. В. Экономный алгоритм выделения блоков в графе.— ЖВМ и МФ, 1974, 14, № 5, с. 1309—1314.
25. Евстигнеев В. А. Теория графов и программирование.— Новосибирск: Изд-во НГУ, 1978.
26. Евстигнеев В. А. Применение бинарных деревьев для организации больших массивов информации (обзор).— В кн.: Прикладная информатика. М.: Финансы и статистика, 1981, вып. 1, с. 170—211.
27. Ермольев Ю. М., Мельник И. М. Экстремальные задачи на графах.— Киев: Наукова думка, 1968.
28. Ершов А. П. О программировании арифметических операторов.— ДАН СССР, 1958, 118, № 3, с. 427—430.
29. Ершов А. П. Сведение задачи распределения памяти при составлении программ к задаче раскраски вершин графов.— ДАН СССР, 1962, 142, № 4, с. 785—787.
30. Ершов А. П. Введение в теоретическое программирование. Беседы о методе.— М.: Наука, 1977.
31. Ершов А. П., Кожухин Г. И. Об оценках хроматического числа связанных графов.— ДАН СССР, 1962, 142, № 2, с. 270—273.
32. Знак В. И. Алгоритм априорной оценки времени машинной реализации программы.— Автометрия, 1972, № 1, с. 101—110.
33. Зыков А. А. Теория конечных графов.— Новосибирск: Наука, 1969.
34. Иорданский М. А. Минимальные нумерации вершин деревьев.— В кн.: Проблемы кибернетики, вып. 31. М.: Наука, 1976, с. 109—132.
35. Иорданский М. А. О плоских размещениях деревьев.— Тезисы докл. IV Всесоюз. конф. по пробл. теорет. киберн. Новосибирск, 1977, с. 137—138.
36. Иорданский М. А. Минимальные плоские размещения деревьев.— В кн.: Методы дискретного анализа в решении

- экстремальных задач. Вып. 33. Новосибирск: ИМ СО АН СССР, 1979, с. 3—30.
37. Иткин В. Э. Характеризация структурированных графов переходов программ.— ДАН СССР, 1980, 250, № 5, с. 1077—1080.
 38. Иткин В. Э. О природе композиционной структурированности программ.— В кн.: Языки и системы программирования. Новосибирск: ВЦ СО АН СССР, 1979, с. 5—20.
 39. Карзанов А. В. Экономный алгоритм нахождения бикомпонент графа.— В кн.: Тр. 3-й Зимней школы по матем. программ. и смежным вопросам, вып. 2. М., 1970, с. 343—347.
 40. Карп Р. М. Заметка о приложении теории графов к программированию для цифровых вычислительных машин.— В кн.: Кибернетический сборник, вып. 4, М.: ИЛ, 1962, с. 123—134.
 41. Касьянов В. Н. Анализ управляющих графов программ.— В кн.: Системное программирование, ч. 2. Новосибирск: ВЦ СО АН СССР, 1973, с. 138—154.
 42. Касьянов В. Н. Об одном алгоритме выделения бикомпонент в ориентированном графе.— В кн.: Системное и теоретическое программирование. Новосибирск: ВЦ СО АН СССР, 1974, с. 235—243.
 43. Касьянов В. Н. Выделение гамаков в ориентированном графе.— ДАН СССР, 1975, 221, № 5, с. 1020—1022.
 44. Касьянов В. Н. Комбинаторные задачи анализа программ в оптимизирующей трансляции: Автореф. дисс. канд. физ.-мат. наук.— Новосибирск, 1975.
 45. Касьянов В. Н. Практический подход к оптимизации программ.— Новосибирск, 1978. (Препринт/ВЦ СО АН СССР: 135).
 46. Касьянов В. Н. Анализ структур программ.— Кибернетика, 1980, № 1, с. 48—61.
 47. Касьянов В. Н. Методы анализа программ.— Новосибирск: Изд-во НГУ, 1982.
 48. Кнут Д. Искусство программирования для ЭВМ. Т. 3: Сортировка и поиск.— М.: Мир, 1978.
 49. Кок Дж. Глобальная экономия команд.— В кн.: Тр. 2-й Всесоюз. конф. по программ. Докл. участников. Новосибирск, 1970.
 50. Корячко В. П., Курчупов В. А. Об укладке графов программ.— Изв. АН СССР. Техническая кибернетика, 1979, № 6, с. 129—136.
 51. Косарев Ю. Г. Распараллеливание по циклам.— В кн.: Вычислительные системы, вып. 24. Новосибирск: ИМ СО АН СССР, 1967, с. 3—20.
 52. Кóфман А. Введение в прикладную комбинаторику.— М.: Наука, 1975.
 53. Кристоффидес Н. Теория графов. Алгоритмический подход.— М.: Мир, 1978.
 54. Лавров С. С. Об экономии памяти в замкнутых операторных схемах.— ЖВМ и МФ, 1961, 1, № 2, с. 309—342.
 55. Лавров С. С., Гончарова Л. И. Автоматическая обработка данных. Хранение информации в памяти ЭВМ.— М.: Наука, 1971.
 56. Липаев В. В. Надежность программного обеспечения АСУ.— М.: Энергия, 1981.

57. Лицаев В. В., Позин Б. А., Блау С. А. Анализ стратегии тестирования программ.— Кибернетика, 1982, № 2, с. 45—50.
58. Мазепа Е. Ю., Федюнькин Е. Д. Оптимальные сети с квазиаддитивной функцией качества.— Дубна, 1980. (Препринт/ОИЯИ: P11-13059).
59. Мазепа Е. Ю., Силин И. Н., Федюнькин Е. Д. О перечислении частичных графов с заданным цикломатическим числом.— Программирование, 1981, № 4, с. 78—86.
60. Мартынюк В. В. Об анализе графа переходов для операторной схемы.— ЖВМ и МФ, 1965, 5, № 2, с. 293—310.
61. Нивергельт Ю., Фаррар Дж., Рейнголд Э. Машинный подход к решению математических задач.— М.: Мир, 1977.
62. Никитин А. С., Венгеров В. Н. Алгоритм субоптимального разрезания взвешенных графов на заданное число блоков.— В кн.: Дискретные преобразования. Киев: ИК АН УССР, 1976, с. 40—50.
63. Оре О. Теория графов.— М.: Наука, 1968.
64. Петросян А. В., Анастасян Ю. Г. Об одной задаче оптимального размещения.— Вопросы радиоэлектроники. Сер. ЭВТ, 1971, вып. 8, с. 80—87.
65. Позин Б. А. Метод структурного построения тестов для отладки управляющих программ.— Программирование, 1980, № 2, с. 62—68.
66. Поиков В. К. Представление графов.— Новосибирск, 1980. (Препринт/ВЦ СО АН СССР: 241).
67. Поиков В. К. Представление деревьев.— Новосибирск, 1981. (Препринт/ВЦ СО АН СССР: 242).
68. Поспелов Д. А. Введение в теорию вычислительных систем.— М.: Сов. радио, 1972.
69. Поттосин И. В. О линеаризации программ и частичном ее упорядочении.— В кн.: Системное и теоретическое программирование. Новосибирск: ВЦ СО АН СССР, 1972, с. 278—286.
70. Поттосин И. В. Оптимизирующие преобразования и их последовательность.— В кн.: Системное программирование, ч. 2. Новосибирск: ВЦ СО АН СССР, 1973, с. 128—137.
71. Реджеёвский Р. Об арифметических выражениях и деревьях.— В кн.: Кибернетический сборник, вып. 7. М.: Мир, 1970, с. 99—107.
72. Рейнголд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы. Теория и практика.— М.: Мир, 1980.
73. Силин И. Н., Федюнькин Е. Д. Итерационные методы оптимизации сети.— Дубна, 1979. (Препринт/ОИЯИ: P5-12995).
74. Тушкина Т. А., Шахбазян К. В. Метод ветвей и границ для задачи параллельного упорядочивания.— В кн.: Записки науч. семинаров Ленингр. отд. МН АН СССР, № 35, 1973, с. 146—155.
75. Уилсон Р. Введение в теорию графов.— М.: Мир, 1977.
76. Фараджев И. А. Алгоритм выделения бикомпонент ориентированного графа.— В кн.: Тр. 3-й Зимней школы по матем. программ. и смежным вопросам, вып. 3. М., 1970, с. 650—654.

77. Филлипович В. В. Об одной задаче преобразования направленного циклического графа в ациклический граф.— Кибернетика, 1973, № 2, с. 138—140.
78. Форд Л. Р., Фалкерсон Д. Р. Потоки в сетях.— М.: Мир, 1965.
79. Хайкин Н. М., Зак Е. И. Построение минимальной укладки дерева в задаче оптимального распределения памяти.— Кибернетика, 1976, № 3, с. 58—63.
80. Харари Ф. Теория графов.— М.: Мир, 1973.
81. Холл М. Комбинаторика.— М.: Мир, 1970.
82. Чебаков В. А. Плоские укладки дерева.— ЖВМ и МФ, 1972, 12, № 2, с. 729—738.
83. Шахбазян К. В., Тухкпина Т. А. О статистической эффективности алгоритмов решения одной задачи разбиения вершин графа.— Кибернетика, 1977, № 5, с. 38—43.
84. Шейдвассер М. А. Об оптимальной нумерации вершин дерева.— В кн.: Дискретный анализ, вып. 17. Новосибирск: ИМ СО АН СССР, 1970, с. 56—74.
85. Шейдвассер М. А. Оптимальное упорядочение вершин ориентированного дерева.— Кибернетика, 1972, № 3, с. 134—139.
86. Шейдвассер М. А. О длине и ширине размещения графов в решетках.— В кн.: Проблемы кибернетики, вып. 29. М.: Наука, 1974, с. 63—102.
87. Штейн М. Е., Штейн Б. Е. Методы машинного проектирования цифровой аппаратуры.— М.: Сов. радио, 1973.
88. Ясикова В. И. Преобразование матрицы смежности ориентированного графа к треугольному виду.— В кн.: Управляемые системы. Вып. 4—5. Новосибирск: Наука, 1970, с. 44—46.
89. Aho A. V., Johnson S. C. Optimal code generation for expression trees.— J. ACM, 1976, 23, № 3, p. 488—501.
90. Aho A. V., Ullman J. D. Node listings for reducible flow graphs.— J. Comp. Syst. Sci., 1976, 13, p. 286—299.
91. Allen B., Munro J. Self-organizing binary search trees.— J. ACM, 1978, 25, № 4, p. 526—535.
92. Allen F. E. Program optimization.— In: Annual Review in Automatic Programming. V. 5, N. Y.: Pergamon Press, 1969, p. 239—308.
93. Allen F. E. Control flow analysis.— SIGPLAN Notices, 1970, 5, № 7, p. 1—19.
94. Alspach B. R., Pullman N. J. Path decompositions of digraphs.— Bull. Austral Math. Soc., 1974, 10, № 3, p. 421—427.
95. Baer J. L. Weight-balanced trees.— In: Proc. AFIPS 1975 NCC. N. J.: AFIPS Press, 1975, p. 467—472.
96. Baer J. L., Caugley R. Segmentation and optimization of programs from cyclic structure analysis.— In: Proc. AFIPS SJCC, v. 40, 1972, p. 23—36.
97. Baker J. J. A note on multiplying Boolean matrices.— Comm. ACM, 1962, 5, № 2, p. 102.
98. Basu S. K. Transformations on directed graphs.— J. Comb. Th., 1970, 9, p. 244—256.
99. Bayer P. J. Improved bounds on the cost of optimum and balanced binary search trees. Techn. Memo. 69. Project Mac. Cambridge: M. I. T., 1975.

100. Bayer R., McCreight C. Organization and maintenance of large ordered indexes.—*Acta Inf.*, 1972, 1, № 3, p. 173.
101. Bien Z. Program graphs.—*Bull. Acad. Pol. Sci. Ser. Sci. math., Astron., Phys.*, 1971, 19, № 8.
102. Bitner J. R. Heuristics that dynamically organize data structures.—*SIAM J. Computing*, 1979, 8, № 1, p. 82—110.
103. Bowie W. S. Applications of graph theory in computer systems.—*Intern. J. of Computer and Inform. Sci.*, 1976, 5, № 1, p. 9—31.
104. Brelaz D. New methods to color the vertices of a graph.—*Comm. ACM*, 1979, 22, № 4, p. 251—256.
105. Bruno B. J., Coffman E. G. Nearly optimal binary search trees.—In: *Proc. IFIP Congress 71. Amsterdam: North Holland Publ. Co.*, 1972, p. 99—103.
106. Chaitin G. J., Auslander M. A. et al. Register allocation via coloring.—*Computer Languages*, 1981, 6, p. 47—51.
107. Cooper D. C. Reduction of programs to a standard form by graph transformation.—In: *Proc. Intern. Seminar on Graph Th. and its Appl.*, Rome, 1966, p. 57—68.
108. Culic K., Ottman Th., Wood D. Dense multiway trees.—*Comp. Sci. Techn. Rep. 79-CS-5.*—Ontario, 1979.
109. Dutton R. D., Bingham R. C. A new graph coloring algorithm.—*The Computer J.*, 1981, 24, № 1, p. 87—91.
110. Earnest C. P., Balke K. G., Anderson J. Analysis of graphs by ordering of nodes.—*J. ACM*, 1972, 19, № 1, p. 23—42.
111. Elshoff J. L., Marcotty M. On use of the cyclomatic number to measure program complexity.—*SIGPLAN Notices*, 1978, 13, № 12, p. 29—40.
112. Estrin G., Turn R. Automatic assignment of computations in a variable structure computer system.—*IEEE Trans. Electron. Comput.*, 1964, EC-12, № 6, p. 755—773.
113. Fredman M. L. Two applications of probabilistic search technique: sorting $x + y$ and building balanced search trees.—In: *7th ACM Symp. on Th. of Comput.* Albuquerque, 1975.
114. Gabow H. N., Macheshwari S. H., Osterweil L. J. On two problems in the generation of program test paths.—*IEEE Trans. on Software Eng.*, 1976, SE-2, № 9, p. 61—64.
115. Garey M. N., Johnson D. S. The complexity of near-optimal graph coloring.—*J. ACM*, 1976, 23, № 1, p. 43—49.
116. Hecht M. S., Ullman J. D. Characterizations of reducible flow graph.—*J. ACM*, 1974, 21, № 3, p. 367—375.
117. Hecht M. S., Ullman J. D. A simple algorithm for global data flow problems.—*SIAM J. Computing*, 1975, 4, № 4, p. 519—532.
118. Hopcroft J. E., Ullman J. D. An $n \log n$ algorithm for detecting reducible graphs.—In: *Proc. 6th Annual Princeton Conf. on Inf. Sciences and Systems*, Princeton, 1972, p. 119.
119. Johnson D. B. Finding all elementary circuits of a directed graph.—*SIAM J. Computing*, 1975, 4, № 1, p. 77—84.
120. Kam J. B., Ullman J. D. Global data flow analysis and iterative algorithms.—*J. ACM*, 1976, 23, № 1, p. 158—171.
121. Karlton P. L., Fuller S. H., Seroggs R. E., Kahler E. B. Performance of height-balanced trees.—*Comm. ACM*, 1976, 19, № 1, p. 23—28.

122. Kildall G. A. An unified approach to global program optimization.— In: Conf. Record of ACM Symp. on Principles of Programming Languages. Boston, Massachusetts, October 1—3, 1974, p. 194—206.
123. Kral J. One way of estimating frequencies of jumps in a program.— Comm. ACM, 1968, 11, № 7, p. 475—480.
124. Kubale M., Dabrowski J. Empirical comparison of efficiency of some graph coloring algorithms.— Arch. automat. i telemekh., 1978, 23, № 1—2.
125. Kwong Y. S., Wood D. *T*-trees: a variant of *B*-trees.— Comp. Sci. Techn. Rep. 78-CS-18.— Ontario, 1978.
126. Leighton F. T. A graph coloring algorithm for large scheduling problems.— J. Res. Nat. Bur. Stand., 1979, 84, № 6, p. 489.
127. Lukes J. A. Efficient algorithm for the partitioning of trees.— IBM J. Res. and Developm., 1974, 18, № 3, p. 217—224.
128. Lukes J. A. Combinatorial solution to the partitioning of general graphs.— IBM J. Res. and Developm., 1975, 19, № 2, p. 170—180.
129. Marcowsky G., Tarjan R. E. Lower bounds on the lengths of node sequences in directed graphs.— Discrete Math., 1976, 16, p. 329—337.
130. Matula D., Marble G., Isaacson J. Graph coloring algorithms.— In: Graph Theory and Computing. N. Y.: Acad. Press, 1972, p. 109.
131. Maurer H. A., Ottman Th., Six H.-W. Implementing dictionaries using binary trees of very small height.— Inf. Proc. Lett., 1976, 5, № 1, p. 11—14.
132. Maurer H. A., Wood D. Zur Manipulation von Zahlenmengen.— Angewandte Informatik, 1976, 18, № 4, S. 143—149.
133. McCabe T. J. A complexity measure.— IEEE Trans. on Software Eng., 1976, SE-2, № 4, p. 308—320.
134. Mehlhorn K. Nearly optimal binary search trees.— Acta Inf., 1975, 5, № 4, p. 287—295.
135. Mezyk S. O pewnej algebraicznej metodzie wyznaczania drog w grafie zorientowanym.— Arch. electrotechn. (PRL), 1973, 22, № 3, s. 701—705.
136. Munro I. Efficient determination of the transitive closure of a directed graph.— Inf. Proc. Lett., 1971, 1, № 2, p. 56—58.
137. Myers G. J. An extension to the cyclomatic measure of program complexity.— SIGPLAN Notices, 1977, 12, № 10, p. 61—64.
138. Nievergelt J., Reingold E. M. Binary search trees of bounded balance.— SIAM J. Computing, 1973, 2, № 1, p. 33—43.
139. Nievergelt J., Wong C. K. On binary search trees.— In: Proc. IFIP Congress 71. Amsterdam: North Holland Publ. Co., 1972, p. 91—98.
140. Nishikawa Y., Yoshida Y., Fukumura T. Top-down algorithms for constructing nearly optimal binary search trees.— Information Processing in Japan, 1977, 17, p. 32—38.
141. Ntafos S. C., Hakimi S. L. On path cover problems in digraphs and applications to program testing.— IEEE Trans. on Software Eng., 1979, SE-5, № 5, p. 520—529.

142. Ottman Th., Six T.-W. Eine neue Klasse von ausgeglichenen Binärbäumen.—Angewandte Informatik, 1976, 18, № 9, p. 395—400.
143. Paige M. R. On partitioning program graphs.—IEEE Trans. on Software Eng., 1977, SE-3, № 6, p. 386—393.
144. Purdom P. A transitive closure algorithm.—BIT, 1970, 10, № 1, p. 76—94.
145. Read R. C., Tarjan R. E. Bounds on backtrack algorithms for listing cycles, paths and spanning trees.—Networks, 1975, 5, № 3, p. 237—252.
146. Sattley K., Millstein R. Comments on a paper by Lowe.—Comm. ACM, 1970, 13, № 7, p. 450—451.
147. Schneck P. B., Angel E. A FORTRAN to FORTRAN optimizing compiler.—The Computer J., 1973, 16, № 4, p. 322—330.
148. Sethi R., Ullman J. D. The generation of optimal code for arithmetic expressions.—J. ACM, 1970, 17, № 4, p. 715—728.
149. Szwarcfiter J. L., Lauter P. E. A search strategy for the elementary cycles of a directed graph.—BIT, 1976, 16, p. 192—204.
150. Tarjan R. E. Depth first search and linear graph algorithms.—SIAM J. Computing, 1972, 1, № 2, p. 146—160.
151. Tarjan R. E. Enumeration of the elementary circuits of a directed graph.—SIAM J. Computing, 1973, 2, № 3, p. 211—216.
152. Tarjan R. E. Testing flow graph reducibility.—J. Comp. Syst. Sci., 1974, 9, № 3, p. 355—365.
153. Tarjan R. Finding dominators in directed graphs.—SIAM J. Computing, 1974, 3, № 1, p. 62—89.
154. Tarjan R. E. Complexity of combinatorial algorithms. Techn. Rep. STAN-CS-77-609.—Stanford, 1977.
155. Thorelli L. E. An algorithm for computing all paths in a graph.—BIT, 1966, 6, p. 347—349.
156. Tiernan J. C. An efficient search algorithm to find the elementary circuits of a graph.—Comm. ACM, 1970, 13, № 12, p. 722—726.
157. Tran Dinh Am, Shuji Tsukiyama et al. An algorithm for generating all the directed paths and its application.—Information Processing in Japan, 1976, 16, p. 31—35.
158. Ullman J. D. Fast algorithms for elimination of common subexpressions.—Acta Inf., 1973, 2, p. 191—213.
159. Unterauer K. Dynamic weighted binary search trees.—Acta Inf., 1979, 11, № 4, p. 340—362.
160. Ver Hoef E. W. Automatic program segmentation based on Boolean connectivity.—In: Proc. AFIPS SJCC, v. 38, 1971, p. 491—495.
161. Warren H. S. A modification of Warshall's algorithm for the transitive closure of binary relations.—Comm. ACM, 1975, 18, p. 218—220.
162. Warshall S. A theorem on Boolean matrices.—J. ACM, 1962, 9, № 1, p. 11—12.
163. Weinblatt H. A new search algorithm for finding the simple cycles of a finite directed graph.—J. ACM, 1972, 19, № 1, p. 43—56.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- АВЛ-дерево 214
 Алгоритм волновой 132
 Аранжировка 39
 Ациклическое упорядочение вершин 97

 Баланс корневой 221
 Бивершина 48
 Бикомпонента 11
 Блок 17
 — корневой 71
 Брат вершины 227

 Величина потока 143
 — разреза 243
 Вершина графа 9
 — безопасная 302
 — ведомая 98
 — ведущая 98
 — висячая 14
 — входная подграфа 12
 — выходная подграфа 12
 — голая 10
 — изолированная 10
 — конечная 10
 — начальная 10
 — подграфа 11
 — отрицательная 146
 — положительная 146
 — разделяющая 17
 — стека 37
 — центральная 14
 — центроидная 15
 Вершины несравнимые 140
 — смежные 9
 — соцветные 288
 Вес вершины 15
 Ветвь блоков 71
 — к вершине 15
 Втягивание вершины 89
 Вход 10
 Выход 10

 Гамак 12
 Грань внешняя 281
 — внутренняя 281
 Граф 12
 — аранжируемый 39
 — бесконтурный 39
 — блоков 17
 — Герца 40, 59
 — двудольный 18
 — двусвязный 17
 — граф информационный 23
 — конденсации 40, 59
 — контурно-связный 118
 — неориентированный 12
 — несепарабельный 17
 — несовместимости 24, 320
 — ориентированный 9
 — полный 9
 — петлевой 118
 — планарный 281
 — плоский 281
 — топологический 15
 — полный 17
 — правильный 145
 — прогрессивно-конечный 273
 — производный 84
 — сводимый 85
 — по Хехту и Ульману 87
 — связный 13
 — сильно связный 11
 — симметричный 9
 — спиральный 97
 — управляющий 19
 — частичный 11
 — k -дольный 283
 — — полный 283
 — k -контурно-связный 118
 — k -раскрасшиваемый 278
 — k -хроматический 278

 Дерево 13
 — ассоциативное 31/
 — балансируемое по весу 221
 — — по высоте 214
 — бинарное 16, 192
 — взвешенное корневое 193
 — доминаторное 164
 — минимаксное 207
 — ориентированное корневое 15
 — поиска в глубину 35
 — сортировки 24
 — упорядоченное 15
 — m -арное 16, 285
 — — r -плотное 236
 Длина графа 167
 — контура 11
 — покрытия 145
 — пути 11
 — укладки 167
 Доминатор 150
 — непосредственный 150
 Допустимое множество функций 121
 Достижимая вершина 11
 Дуга 9

Дуга безопасная 302

- древесная 36
- обратная 33
- отмеченная 34
- поперечная 36
- прямая 36

Зона 11

- Замыкание 63
- обратное 63
- транзитивное 53

Иерархия зон 64

- Изолированное множество 258
- Интервал 83
- Интервальное представление 84

Каркас 16

- Клива 17
- Код Прюфера 29
- Харари 28
- Композиция управляющих графов 298
- Компонента линейная 12
- сильной связности 11
- Конец дуги 9
- Контур 11
- гамильтонов 11
- простой 11
- Эйлеров 11
- Кратность покрытия 145

Маршрут величины 320

- Матрица вложенности 113
- достижимости 19
- инцидентности 19
- смежности 18
- фундаментальных циклов 117
- цикломатическая 113
- Мультиграф 13
- ориентированный 9

Начало дуги 9

- Носитель орграфа 88
- Нумерация вершин 33
- базисная 34
- допустимая 166
- линейная 34
- оптимальная 166
- плоская 166
- прямая 41

Область действия величины 320

- с головной вершиной 96
- Образующая разрезания 260
- Объединение управляющих графов 296
- Обязательный приемник 82
- Одноцветный класс 278
- Окрестность вершин 288
- Операторная схема 23
- Орграф 9
- сильно связный 11
- симметричный 9
- Оркаркас 17
- Остов 16

Паросочетание 143

- максимальное 143
- Петля 9
- Подграф 11
- Поиск в глубину 34
- Покрытие, путевое 139
- — минимальное 140
- — приведенное 145
- Полугамак 82
- Позурешетка 121
- ограниченная 122
- Полустепень захода 10
- исхода 10
- Поток 142
- минимальный 142
- Потомок вершины 10
- Предок вершины 10
- Предшественник вершины 10
- Путь 10
- вырожденный 11
- гамильтонов 11
- кратчайший 130
- критический 133
- отмеченный 216
- простой 10
- текущий 34
- транслирующий 149
- тривиальный 11
- Эйлеров 11

Радиус графа 14

- Разметка вершин 123
- Разрез 242
- минимальный 243
- собственный 243
- Разрезание доминантное 239
- допустимое 234
- подобное 250
- Ранг контурный графа 118
- Раскраска последовательная 283
- Растянутасть укладки 166
- Ребро 12
- Редукционный порядок 94

Связное множество 254

- Семидоминатор 154
- Сечение 118
- минимальное 118
- Сжатие укладки 166
- Склеивание вершин 288
- Склейка 134
- Слабая последовательность вершин 95
- Смежные вершины 9, 13
- Стек 37
- Степень вершины 10, 13
- Стоимость дерева 193
- Строгая последовательность вершин 95
- Суграф 11

Тест 305

- Топологическая сортировка вершин 60
- Точка сочленения 17, 70
- Транзитивное замыкание 53
- Трудоемкость алгоритма 31

Укладка 166
— оптимальная по длине 167
— — по числу соседней связности 190
— — по ширине 167
Уровень вершины 193

Фактор-граф 53
Фокус вершины 82

Хорда 116
Хроматическое число 278

Центр 14
Центронд 15
Цепь 13
— полурешетки 122
Цикл 13
— формальный 41
— — примитивный 41
— — самый внутренний 65
— фундаментальный 117
Цикломатическая сложность про-
граммы 304
Цикломатическое число 116

Число соседней связности 190

Ширина графа 167
— плоской укладки 170

Ширина укладки 167
Шлейф вершины 82

Эксцентриситет 14

A -нумерация 39
 B -дерево 330
 C -удаление вершины 302
 F -достижимость 34
 F -линия 50
 F -область 47
 F -ранг 50
 F -путь 34
 $f(m)$ -дерево 335
 H -дерево 226
 HB -дерево 227
 HS -дерево 230
 h -дерево 231
 K -нумерация 49
 k -раскраска правильная 278
 k -смежность 254
 KLC -граф 118
 L -нумерация 50
 LC -граф 118
 M -нумерация 35
 N -нумерация 38
 P -нумерация 41
 $s - t$ -путь 10
 T -дерево 338
 T -нумерация 48
 λ -граф 213
2-3-дерево 331